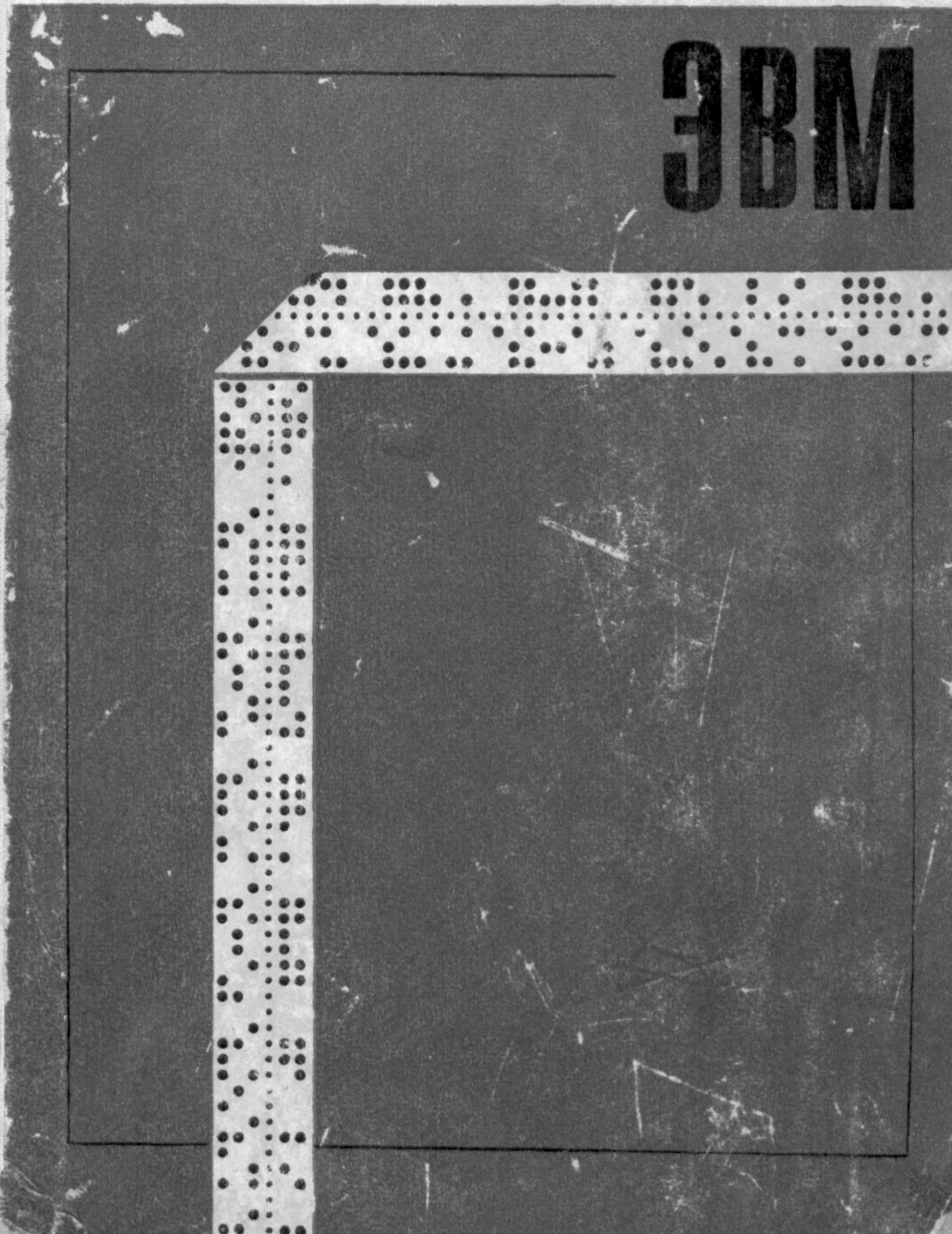
З.В. Алферова
Г.Н. Лихачева
В.В. Шураков

Математическое обеспечение ЭВМ



З. В. АЛФЕРОВА, Г. Н. ЛИХАЧЕВА, В. В. ШУРАКОВ

МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ЭВМ

Допущено Министерством высшего и среднего специального образования СССР в качестве учебного пособия для студентов вузов, обучающихся по специальности «Организация механизированной обработки экономической информации»

МОСКВА «СТАТИСТИКА» 1974

Учебное пособие представляет собой опыт обобщения работ в области проектирования математического обеспечения (МО) ЭВМ. В основном использовались концепции и методы разработки МО для вычислительных систем третьего поколения.

Книга состоит из пяти разделов, освещающих математические модели режимов работы операционных систем, концепции систем программирования, вопросы проектирования операционных систем, компиляторов, пакетов прикладных программ, испытательных программ. Материал не привязывается к разработке МО конкретных вычислительных систем и является обобщением отечественного и зарубежного опыта такого рода работ.

Учебное пособие предназначено для студентов, но будет полезно аспирантам, системным программистам, проектировщикам автоматизированных систем управления и другим специалистам, занимающимся вопросами общего или специального математического обеспечения.

А 30502—082 87—74
008(01)—74

ПРЕДИСЛОВИЕ

Эффективная эксплуатация современных электронных вычислительных машин (ЭВМ) не мыслится без оснащения их комплектом специальных программ, облегчающих процесс подготовки задач к решению и организующих прохождение этих задач через машину. Комплекс таких программ принято называть программным или математическим обеспечением. В современных вычислительных системах мощные и эффективные средства программного обеспечения осуществляют распределение ресурсов, координацию работ, загрузку программ, вывод информации, диагностику ошибок и самые различные функции управления.

Усложнение архитектуры вычислительных систем приводит к изменению и усложнению программного обеспечения. Комплекс программ современных вычислительных машин составляет более 1,5 млн. команд.

Авторы не ставили перед собой цели дать детальное описание программного обеспечения, а постарались изложить основы программного обеспечения, не привязываясь к конкретным вычислительным системам, по возможности обобщая основные идеи разработки программного обеспечения в СССР и за рубежом.

Структурное построение работы по разделам соответствует курсам лекций, читаемых авторами в Московском экономико-статистическом институте.

Первый раздел посвящен характеристике программного обеспечения ЭВМ и включает вопросы развития вычислительной техники и программного обеспечения, особенности функционирования вычислительных машин третьего поколения и режимы эксплуатации систем. В этом разделе приводится классификация программного обеспечения, принципы его проектирования и документирования.

Второй раздел посвящен испытательным программам контроля и диагностики машин. В этом разделе содержатся сведения по проектированию испытательных программ и приведены некоторые критерии оценки их качества.

Третий раздел работы посвящается системам программирования и включает описание конструкции таких компонентов программного обеспечения, как ассемблеры, интерпретаторы и компиляторы. В этом же разделе излагаются вопросы разработки трансляторов.

При создании и эксплуатации систем программного обеспечения возникает много проблем теоретического и инженерного характера,

среди которых главенствующее место занимает построение операционной системы. Этим вопросам и посвящен четвертый раздел работы. Здесь излагаются общие методы построения операционных систем.

В пятом разделе работы рассматриваются основные принципы разработки пакетов прикладных программ на нескольких примерах.

Авторы пытались применять в своей работе наиболее широко используемые термины в вышедшей литературе по вычислительным системам и программному обеспечению.

С целью удобства чтения книги список использованной литературы представлен по каждому разделу отдельно.

При изучении данного курса необходимы знания основ вычислительной техники, алгоритмических языков, программирования для конкретных ЭВМ, математической статистики и исследования операций.

В конце работы для удобства пользования приводится предметный указатель.

РАЗДЕЛ 1

ХАРАКТЕРИСТИКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ЭВМ

Глава 1

ЭТАПЫ РАЗВИТИЯ ЭВМ И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1.1. Развитие вычислительной техники

В настоящее время вычислительная техника интенсивно внедряется во все сферы деятельности человека. Гибкое программное управление — главное, что отличает современные электронные машины от вычислительной перфорационной техники, являющейся предшественницей этих машин, и новая техническая революция, проходящая на наших глазах, во многом обязана появлению ЭВМ.

С рождением вычислительной техники связаны имена Чарльза Бэббиджа, Джорджа Буля и Германа Холлерита.

Чарльзу Бэббиджу принадлежит честь формулировки принципа программного управления. Бэббидж — профессор математики Кембриджского университета (Англия) — изобрел в 1812 г. разностную вычислительную машину. Эта машина могла выполнять вычисление таблиц функций, заданных полиномом n-й степени. Метод основывался на том, что у полиномиальной функции n-я разность постоянна, и если заданы начальные значения функции и ее разности n-го порядка, то несложными вычислениями можно построить таблицу функций на любом интервале.

Бэббидж построил машину, позволяющую табулировать функции — полиномы второго порядка, и разработал проект машины для табулирования полиномов шестого порядка, которая не была построена. Бэббиджем были сформулированы принципы программного управления и запоминаемой программы, которые осуществлены в современных ЭВМ.

В 1854 г. английский математик Джордж Буль опубликовал книгу «Законы мышления», в которой развил алгебру высказываний, получившую затем название Булевой алгебры. На основе Булевой алгебры в начале нашего столетия развивались теория релейно-контактных схем и конструирование сложных дискретных автоматов.

Многогранно влияние алгебры высказываний на развитие вычислительной техники: она была инструментом разработки и анализа сложных схем, инструментом оптимизации числа логических элементов, из многих тысяч которых состоит современная ЭВМ.

Отцом вычислительной перфорационной техники, предшественницы современных машин, часто называют американского ученого Холлерита, который, занимаясь вопросами обработки данных переписи населения в восьмидесятых годах прошлого столетия, изобрел машину обработки данных переписи и перфокарту — носитель этой информации. Машина Холлерита явилась прообразом современных табуляторов и вычислительных перфорационных устройств и использовалась при обработке данных переписи 1890 года. В 1896 г. Холлеритом была основана фирма по выпуску вычислительных перфорационных машин и перфокарт. Эта фирма затем была преобразована в фирму ИБМ, являющуюся ныне самым крупным поставщиком электронных машин во всем мире.

В 1937 г. американец Говард Айткен из Гарвардского университета предложил проект большой релейной машины, содержащей 72 сумматора и 60 тумблерных регистров для набора констант. Программа работы машины набиралась на коммутационных досках и переключателях. Машина была построена фирмой ИБМ в 1944 г. и названа МАРК-1. Эта машина еще не обладала гибко изменяющейся программой вычислений, но доказывала реальность проекта построения большой машины, состоящей из тысячи элементов.

Первой электронной машиной принято считать ЕНИАК, которая была разработана сотрудниками Пенсильванского университета и построена в 1945 г. Эта ламповая машина уже обладала автоматическим программным управлением, но не имела внутреннего запоминающего устройства.

Первая машина, обладающая всеми необходимыми компонентами современных ЭВМ, была построена в 1949 г. в Англии и названа *EDSAC*.

В Советском Союзе в 1947 г. коллектив работников под руководством академика С. А. Лебедева начал разработку малой электронной машины МЭСМ, которая вступила в строй в 1951 г. Таким образом, в начале 50-х годов наступил период бурного развития вычислительной техники как у нас в стране, так и за рубежом. В 1953 г. в Советском Союзе начался серийный выпуск машин, первыми из которых были «БЭСМ-1» и «Стрела». В США первые серийные машины появились в 1951 г. (две машины модели УНИВАК). Кроме того, фирма ИБМ в это же время начала серийный выпуск машины ИБМ/701. Если в 1952—1953 гг. насчитывалось несколько десятков электронных машин, то в 1965 году их было уже около 40 тысяч, а в 1970 г. — свыше 100 тысяч. Эти цифры характеризуют темпы развития вычислительной техники за 20 с небольшим лет ее существования.

Принято различать четыре поколения в развитии электронных вычислительных машин.

Первое поколение машин — ламповые машины с быстродействием порядка 10—20 тыс. операций в секунду, которые использовались в основном для научных расчетов. Программы для этих машин писались вручную.

Второе поколение — транзисторные машины, или машины на твердых схемах с максимальным быстродействием порядка сотен тысяч операций в секунду. Для этих машин имелись достаточно полно разработанные библиотеки программ и трансляторы с некоторых алгоритмических языков.

Третье поколение — машины на интегральных схемах и вычислительные системы с максимальным быстродействием порядка миллиона операций в секунду. Вычислительные системы третьего поколения представляют собой семейство машин общего назначения, разработанных на единой структурной и микроэлектронной базе.

Первые образцы машин третьего поколения, созданные американской фирмой International Business Machines Corporation, появились во второй половине 1965 г. и были названы семейством машин ИБМ/360. Вычислительная система ИБМ/360 насчитывает свыше семи моделей, из которых наиболее простая примерно в 30 раз дешевле самой мощной из них. Принципы построения системы ИБМ/360 оказали существенное влияние на многие крупные фирмы, производящие вычислительные машины. Такие фирмы, как *SDS*, *RCA* (США), *ICL* (Англия), *SIEMENS* (ФРГ), *BULL* (Франция), *OLIVETTI* (Италия), *HITACHI* (Япония), выпускают машины полностью или в значительной мере совместимые с ИБМ/360. В СССР в настоящее время выпускается Единая Система Электронных Вычислительных Машин ЕС ЭВМ, в значительной части совместимая с ИБМ/360.

Четвертое поколение — многопроцессорные машины на больших интегральных схемах (БИС), обеспечивающие быстродействие в десятки миллионов операций в секунду. Эти машины должны обеспечивать коммунальное использование вычислительных мощностей на базе объединения в единую вычислительную сеть.

Бурный рост количества ЭВМ сопровождался также ростом их эксплуатационных возможностей. Ведущими фирмами анонсированы новые семейства ЭВМ или добавлены новые модели в существующие семейства. Так, фирма ИБМ разработала новое семейство ЭВМ ИБМ/370, представленное моделями средних возможностей 145, 155 и 165. Новая серия ЭВМ не относится к четвертому поколению, но при ее разработке были учтены все предъявляемые к современным ЭВМ требования: возможность работы в мультипрограммном режиме, в системах с разделением времени, возможность использования для дистанционной пакетной обработки данных. Новая серия разработана на основе принципа преемственности, который выразился в совместимости и взаимозаменяемости аппаратуры, а также средств математического обеспечения серии ИБМ/360.

Большое количество новых машин было создано другими фирмами. Например, фирма *RCA* разработала семейство машин сред-

них возможностей *RCA*-2, 3, 6, 7, фирма *Burroughs* приступила к разработке семейства *Burroughs*-700, включающего средние ЭВМ *B*-5700, *B*-6700 и большую ЭВМ *B*-7700. Новые машины выгодно отличаются от предшествующих, так как в них были использованы более быстродействующие и меньшие по габаритам логические схемы, более сложная организация памяти при большей емкости и высоком быстродействии. Так, в Иллинейском университете завершается разработка и создание супермашины *ILLIAC* IV с быстродействием порядка 200 млн. опер/сек.

Наравне с супермашинами в настоящее время выпускаются мини-машины самого различного назначения. Мощность современной мини-машины третьего поколения примерно эквивалентна мощности средней или даже большой ЭВМ первого поколения.

Предполагается, что минимашина четвертого поколения должна сочетать в себе пять основных качеств: возможность микропрограммирования, модульную конструкцию, невысокую стоимость, применимость в соответствии с требованиями пользователя и минимальные сроки поставки.

Основными тенденциями конструирования ЭВМ продолжают оставаться обеспечение надежности работы аппаратуры, ее компактности и баланс между экономической эффективностью и производительностью.

1.2. Развитие средств программного обеспечения

С ростом сложности архитектуры ЭВМ и повышением уровня автоматизации их работы все большее значение приобретают вопросы программного обеспечения ЭВМ. Доля неаппаратурной части в стоимости ЭВМ третьего поколения уже сейчас превышает по зарубежным источникам 50%.

Одновременно с появлением цифровых программно-управляемых машин возникла новая область прикладной математики — программирование.

Программирование как область науки и как профессия возникла в 50-х годах нашего века. Значительный отрезок времени своего существования программирование представлялось скорее искусством, чем наукой, так как составление программ производилось вручную и было подобно решению сложных комбинационных задач с большими затратами ресурсов.

Принято выделять три этапа в развитии автоматизации программирования и организации прохождения программ через машину. В различных странах содержание и последовательность разработки средств автоматизации программирования имеет свои особенности, но общая тенденция при этом сохраняется.

Появление в 1953 г. операторного метода программирования, разработанного А. А. Ляпуновым, позволило по-настоящему поставить вопрос об автоматизации программирования. Сущность операторного метода заключалась в том, что алгоритм решения задачи

представлялся в виде совокупности операторов, образующих логическую схему задачи. Логические схемы задач позволяли расчленить громоздкий процесс составления программы и допускали составление по формальным правилам программ этих частей с последующим объединением их в одно целое. На идеях операторного метода программирования и были основаны первые программирующие программы (ПП).

В 1954 г. в СССР была разработана первая программирующая программа ПП-1, явившаяся макетом для проверки идей операторного метода программирования. На базе ПП-1 в 1955 г. была разработана более совершенная ПП-2 и др.

В США с возникновением в 1954 г. алгебраического подхода к описанию программ, по существу совпадающему с операторным методом, также стал развиваться метод программирующих программ. В 1956 г. корпорацией ИБМ была разработана мощная универсальная ПП «ФОРТРАН» для автоматического программирования на наиболее массовой в то время машине ИБМ/704. К концу 50-х годов метод ПП прочно вошел в практику программирования.

Наряду с развитием метода программирующих программ возникали и совершенствовались приемы автоматизации отдельных работ ручного программирования, например отладки и редактирования программ.

Идеи операторного метода легли в основу создания алгоритмических языков описания процессов обработки информации. Алгоритмический язык представляет собой набор символов и правил или соглашений, управляющих способом и последовательностью, в которой символы могут соединяться в осмысленное сообщение. Программы, написанные на алгоритмическом языке, при наличии специальных программ (трансляторов) могут быть переведены с помощью ЭВМ на ее конкретный машинный язык.

Независимость алгоритмического языка от ЭВМ позволяет вовлекать в сферу алгоритмизации задач специалистов различных отраслей знаний, не являющихся вычислителями, стандартизировать типовые программы, устранять дублирование в написании программ на различные типы ЭВМ и ускорить процесс прохождения задач от постановки до получения результатов.

На начало 70-х годов в мире имелось свыше 700 алгоритмических языков и около 300 трансляторов для автоматизации программирования. В практике программирования продолжают доминировать алгоритмические языки КОБОЛ, ФОРТРАН, АЛГОЛ-60, ЛИСП, ПЛ-1.

Наиболее серьезной разработкой последних лет является создание универсального алгоритмического языка АЛГОЛ-68, для которого в настоящее время разрабатываются трансляторы как у нас в стране, так и за рубежом.

Анализ разработок языков последних лет показывает, что темпы развития языков существенно замедляются. Это замедление в развитии совсем не означает, что проблема исчерпана. В зарубежных

источниках высказывается мнение о том, что для правильного понимания и выбора должного направления в разработке языков необходимо построить новые разрезы, по которым различаются семейства языков. Считается, что сейчас необходимы языки для начинающих, для опытных программистов, для профессиональных программистов. Выделяется группа языков для непрофессионалов: учащихся, врачей, бухгалтеров и других устойчивых профессиональных групп.

Большой интерес представляют сводные данные об использовании средств программирования в 1968 г. на машинах, принадлежащих государственным учреждениям США (табл. 1.1).

Дальнейшее усложнение структуры ЭВМ привело к созданию специальных управляющих программ, обеспечивающих организацию и прохождение задач через машины. Основные идеи операционной системы (ОС) возникли в 1953 г., когда были созданы первые управляющие системы для машин второго поколения. К ним относится система МТИ для учебных приложений, разработанная в Массачусетском технологическом институте. Основным ее назначением было обеспечение непрерывного прохождения через машину совокупности многих работ и предоставление возможности пользоваться хранимой в машине библиотекой служебных программ. Причиной возникновения такого рода систем послужило желание совместить операции по запуску (установка носителей и т. д.) с выполнением программ. При этом работы выполнялись последовательно, будучи выбранными из группы работ. Такие работы в дальнейшем получили название пакета работ. Последовательное выполнение работ с помощью управляющей программы получило название пакетной обработки.

Таблица 1.1

Группы и наименование средств программирования	В процентах к итогу в группе	В процентах к общему итогу
Проблемно-ориентированные языки	100	42
в том числе:		
КОБОЛ	69	29
ФОРТРАН	28	12
ПЛ/1	3	1
Машинно-ориентированные языки	100	58
в том числе:		
ассемблеры	83	48
генераторы отчетов	16	9
машинные языки	1	1

Операционные системы, выполняющие пакетную обработку, относятся к первому поколению ОС. Для них характерно, что в основной памяти находится резидентная часть операционной системы.

Вся остальная часть и другие ресурсы машины отдавались в распоряжение одной задаче независимо от того, использовала она их с полной нагрузкой или нет.

Параллельно с развитием пакетной обработки были созданы другие типы операционных систем. Эти системы осуществляли полное подчинение требуемых ресурсов машины одной работе, выполняемой в «реальном времени». Другие работы, выполняемые в это время, ожидали освобождения ресурсов и исполнялись последовательно, когда ресурсы освобождались. Представителями систем такого рода являются операционные системы *SAGE, MERCURY, SABRE*. Данные системы, хотя и относятся к первому поколению, мало похожи на системы пакетной обработки как с точки зрения применения, так и с точки зрения структуры. Они также разрабатывались для машин второго поколения.

Из ОС первого поколения, разработанных в нашей стране, можно назвать мониторную систему ФОРТРАН, созданную для ЭВМ «Минск-22», операционные системы, созданные для машин «Минск-32», БЭСМ-6 и др.

Операционными системами первого поколения, созданными фирмой ИБМ совместно с пользователями, входящими в ассоциацию *SHARE*, были *SOS* (*SHARE Operating System*) и *FMS* — мониторная система ФОРТРАН (*FORTRAN Monitor System*). Первые операционные системы второго поколения были также разработаны фирмой ИБМ. Отличительной чертой является то, что они обеспечивают и пакетную обработку, и работу в реальном времени. Более поздние операционные системы второго поколения обеспечивают работу мультипроцессорных систем, систем с разделением времени. Эти операционные системы созданы для машин третьего поколения.

В апреле 1964 г. фирма ИБМ объявила о создании большой модели из семейства машин ИБМ/360. В декабре 1965 г. 30-я модель была установлена в Лос-Анжелосе, а в конце января 1966 г. были отлажены основные управляющие программы и компилятор с языка КОБОЛ. В настоящее время операционная система ОС/360 насчитывает около 1,6 млн. команд.

Из операционных систем, созданных другими фирмами, следует отметить систему Джей, созданную английской фирмой *ICL*, управляющую систему *SIEMENS*-4004 (ФРГ), Джикестрин для *GENERAL ELECTRIC*, созданную итальянскими и американскими специалистами, операционную систему РОБОТРОН (ГДР) и др. В 1972 г. была разработана дисковая операционная система Единой системы электронных вычислительных машин (СССР).

Современное программное обеспечение для семейства ЭВМ отличается сложностью структуры и может включать операционную систему, различные языки и компиляторы с них, тестовые программы контроля и диагностики, набор обслуживающих программ.

Японские специалисты выдвинули интересную идею проектирования системы программного обеспечения с «кольцевой структурой» (рис. 1.1).

Продолжением аппаратных возможностей машины является ядро операционной системы. Эта область зависит от аппаратуры и должна быть «привязана» к спецификациям на аппаратуру. По мере удаления от центра зависимость программного обеспечения от аппаратуры уменьшается. Для связи с аппаратурой и ядром операционной системы независящие части программного обеспечения используют возможности системы программ сопряжения, включающие, в частности, методы макропрограммирования, специальные языки для описания компиляторов и т. п. Самую внешнюю оболочку составляют программы пользователей. Процесс проектирования такой системы предусматривает предварительную разработку программного обеспечения, зависящего от аппаратуры, с последующим его использованием в целях проектирования и разработки программного обеспечения, независящего от аппаратуры.

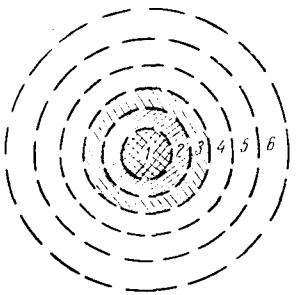


Рис. 1.1. Кольцевая структура ЭВМ:

1 — аппаратура; *2* — ядро операционной системы; *3* — программы сопряжения; *4* — часть ОС, ориентированная на пользователя; *5* — системы программирования; *6* — программы пользователей

1.3. Стандартизация программного обеспечения

Производительность ЭВМ в равной мере определяется как техническими характеристиками ее оборудования (быстродействием, объемом памяти, надежностью и т. д.), так и составом и качеством применяемого при эксплуатации программного обеспечения (ПО).

Серийный выпуск ЭВМ с достаточно полным комплектом программ, последовательное расширение этого комплекта у пользователей за счет дополнительной поставки новых программ заводом-поставщиком и обмена апробированными алгоритмами и программами между пользователями позволяет значительно уменьшить трудоемкость и стоимость обработки данных. При этом создаются благоприятные условия для использования как отдельной машины, так и парка ЭВМ страны в целом.

Эти проблемы могут быть успешно решены лишь на основе унификации и стандартизации элементов программного обеспечения и комплексов ЭВМ, а также введения единых требований к процессу обмена информацией по программному обеспечению.

К числу основных объектов стандартизации относятся:

1) обозначения и терминология по программному обеспечению ЭВМ и специализированных систем обработки данных;
2) алгоритмические языки;
3) алгоритмы;
4) программы и модули программ решения задач на ЭВМ;
5) системы программного обеспечения и их компоненты;
6) организация процесса разработки и внедрения ПО ЭВМ.

Унификация состава ПО ЭВМ и регламентация этапов его разработки с учетом технических возможностей и особенностей применения машин являются одним из важнейших средств урегулирования интересов производителей и пользователей ЭВМ.

В нашей стране первым шагом на пути унификации ПО ЭВМ было утверждение Государственным комитетом по координации научно-исследовательских работ в 1965 г. положения «Типовой состав систем математического обеспечения ЭВМ». Этим положением устанавливался минимальный перечень программ и методических материалов, входящих в состав эксплуатируемых, серийно выпускаемых и разрабатываемых отечественных ЭВМ.

За рубежом также активно проводилась и проводится работа по стандартизации программного обеспечения ЭВМ. Имеющиеся в настоящее время стандарты включают:

обозначения, в том числе и обозначения блок-схем для программирования на ЭВМ (Англия, США, ЧССР, Франция);

терминологию, в том числе терминологию в области обработки данных (Англия, США, ЧССР, Япония);

алгоритмические языки (США, Франция).

Три направления национальной стандартизации ПО ЭВМ, перечисленные выше, являются основными и для международной стандартизации в этой области.

В целях улучшения организации вычислительных работ и снижения трудоемкости подготовки задач к решению на ЭВМ путем организованного накопления и распространения алгоритмов и программ Государственный комитет Совета Министров СССР по науке и технике Постановлением от 21.02.1966 г. принял решение о создании и развитии Государственного фонда алгоритмов и программ (ГФАП) и утвердил Положение о порядке подготовки, апробации и представления материалов в ГФАП, а также Инструкцию по оформлению представляемых в фонд алгоритмов, программ и инструктивно-методических материалов.

Согласно этому Положению ГФАП является частью Генерального справочно-информационного фонда страны по естественным и техническим наукам и включает соответствующие материалы фондов Государственной публичной научно-технической библиотеки (ГПНТБ) СССР, всесоюзных и центральных отраслевых органов информации, а также организаций и предприятий, использующих ЭВМ.

В фонд включаются:

1) алгоритмы и программы для решения научно-технических и экономических задач;

2) программы, входящие в системы ПО различных типов ЭВМ;

3) методические и инструктивные материалы по программированию, алгоритмическим языкам и организации вычислительных работ на ЭВМ;

4) информационные и справочно-библиографические материалы по алгоритмам, программам и системам ПО.

Работы по накоплению, хранению, систематизации, размножению и рассылке материалов заинтересованным организациям возложены на центральные отраслевые органы научно-технической информации.

Ответственность за подготовку, апробацию, оформление и поступление материалов в отраслевые (ведомственные) фонды алгоритмов и программ возложена на ведущие организации, определяемые министерствами, государственными комитетами и ведомствами.

Разработка вопросов методологии информационной работы по программному обеспечению ЭВМ, ведение всесоюзной картотеки алгоритмов и программ, издание информационных и методических сборников материалов по ПО ЭВМ возложены на Научно-методический центр по программному обеспечению ЭВМ при ВЦ АН СССР.

Установлен следующий порядок поступления материалов в Госфонд:

1) материал (алгоритм, программа и т. д. в двух экземплярах) оформляется разработчиком в соответствии с инструкцией;

2) организация-разработчик направляет материал в соответствующий отраслевой (ведомственный) орган научно-технической информации, а также хранит ~~их~~ 2хр в собственном фонде;

3) ведущие организации отрасли апробируют материал и дают заключение о целесообразности включения их в отраслевой фонд или о доработке;

4) на основе заключения ведущей организации центральные отраслевые органы научно-технической информации организуют распространение материалов заинтересованным организациям и направляют один экземпляр информационных карт в научно-методический центр;

5) при издании отраслевыми фондами три экземпляра их направляются в ГПНТБ СССР.

Для научно-методического руководства проводимыми в стране работами по ПО ЭВМ образована Межведомственная научно-техническая комиссия. В ее составе создана постоянная секция по ГФАП, призванная координировать работы по развитию и совершенствованию Госфонда.

Действующая Инструкция по оформлению представляемых в ГФАП материалов требует, чтобы вместе с ними представлялась информационная карта, содержащая основные сведения о разработчике (наименование предприятия, организации с указанием ведомственной подчиненности, адрес и телефон, дата выполнения работы и авторы работы), библиографические данные и аннотацию на материал.

Аннотации на алгоритмы и программы должны давать полную информацию о возможности их использования, аннотация на инструктивно-методические материалы должна раскрывать их содержание и назначение.

Нормальное функционирование ГФАП немыслимо без стройной

системы классификации. В ВЦ АН СССР при участии ряда организаций был разработан Классификатор Государственного фонда алгоритмов и программ.

Основные направления работ по унификации и стандартизации ПО ЭВМ, на которых целесообразно сосредоточить внимание, следующие:

1) разработка типового состава систем ПО ЭВМ, соответствующего современному состоянию вычислительной техники и условиям ее использования для решения конкретных типов экономических и научно-технических задач;

2) разработка рекомендаций по специальному программному обеспечению, предназначенному для эффективного использования ЭВМ в специализированных системах обработки данных;

3) стандартизация алгоритмических языков, алгоритмов и программ;

4) разработка отраслевых, ведомственных и общегосударственных руководящих технических материалов, унифицирующих порядок разработки и развития систем ПО, устанавливающих технико-экономические показатели системы и ее элементов, определяющих нормативы на разработку и внедрение систем и обмен информаций в этой области;

5) унификация терминологии и обозначений в области ПО ЭВМ и специализированных систем обработки данных.

Стандартизация ПО ЭВМ должна проводиться с учетом тенденций развития вычислительной техники и, в свою очередь, создавать необходимые предпосылки для ускорения темпов разработки и внедрения ЭВМ, способствовать повышению качества обработки информации. Системный подход и преемственность в разработке и развитии ПО ЭВМ позволяют сократить сроки и трудоемкость разработки новых моделей ЭВМ, создать условия для ускорения и упрощения процесса программирования, а также уменьшения стоимости решения задач на ЭВМ.

Основой подобного системного подхода является унификация средств и методов программирования, библиотек программ и других элементов ПО ЭВМ. Такая стандартизация должна не только обеспечивать высокий уровень программного обеспечения отдельных семейств машин одного поколения, но и создавать предпосылки для наращивания ПО новых типов ЭВМ и следующих поколений, т. е. в этом смысле быть опережающей.

Формы стандартизации должны быть достаточно гибкими, должны способствовать своевременному формированию общих требований и достаточно полной их апробации. Целесообразно наряду со стандартами на программное обеспечение ЭВМ разрабатывать и конкретные рекомендации по унификации.

Существуют большие преимущества в проведении этих мероприятий по стандартизации при социалистической системе планирования и управления, позволяющей добиться значительного технико-экономического эффекта за счет осуществления единой технической

политики в области развития средств вычислительной техники, программного обеспечения и рационального использования всего парка машин страны.

ВОПРОСЫ К ГЛАВЕ

1. Перечислите основные тенденции в развитии вычислительной техники в СССР и за рубежом.
2. Чем отличаются ЭВМ первого, второго и третьего поколения?
3. Назовите этапы развития автоматизации программирования.
4. Основная тенденция развития операционных систем.
5. Назначение Государственного фонда алгоритмов и программ.
6. Порядок поступления материалов в Государственный фонд алгоритмов и программ.
7. Основные направления работ по стандартизации и унификации программного обеспечения.

Глава 2

ОСОБЕННОСТИ ФУНКЦИОНИРОВАНИЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ ТРЕТЬЕГО ПОКОЛЕНИЯ

2.1. Общая характеристика вычислительных систем третьего поколения

Вычислительные системы третьего поколения предназначены для использования во всех областях обработки информации. Они дают возможность обрабатывать большие архивы данных, решать технические, научные и экономические проблемы как отдельно, так и одновременно.

Стандартность сопряжения устройств ввода-вывода с каналами ввода-вывода, стандартность набора команд и кодирования данных позволяют пользователю одной системы использовать программное обеспечение, созданное для другой аналогичной системы, и позволяют легко переходить от одной системы к другой.

Все модели одной вычислительной системы третьего поколения считаются совместимыми «вверх» и «вниз». Совместимость «вверх» означает, что программа, написанная для младшей модели, должна выполняться на старшей модели и давать правильный результат. Совместимость «вниз» означает, что программа, написанная для старшей модели, должна выполняться на младшей модели и тоже давать правильный результат. Младшей моделью системы будем считать машину с минимальной конфигурацией. Под конфигурацией машины будем понимать имеющийся в ней набор внешних устройств ввода-вывода, набор команд, поставляемый объем памяти, скорость выполнения команд процессором. Следовательно, более старшая модель системы будет отличаться от младшей бо́льшим количеством и разнообразием устройств ввода-вывода, расширен-

ным набором команд, бо́льшим объемом оперативной памяти, быстродействием центрального процессора.

Совместимость «вниз» предполагает ряд существенных ограничений: возможности моделей по отношению к программе должны в каждом случае быть одинаковыми, т. е. центральные процессоры должны обладать одинаковыми наборами необязательных (поставляемых по выбору заказчика) возможностей; минимальная память должна иметь одинаковую структуру; минимальное количество, тип и приоритет устройств ввода-вывода должны быть эквивалентными; программа не должна зависеть от времени выполнения команд центральным процессором, скорости ввода-вывода данных, времени выполнения команд каналами ввода-вывода и др.

Эти ограничения практически приводят к несовместимости «вниз», от старших моделей к младшим, так как в противном случае программы не должны быть написанными с использованием всех возможностей, которые им предоставляют старшие модели. Совместимость «вверх» является обязательным свойством, обеспечивающим возможность наращивать мощности моделей без изменения существующего задела программ.

Вычислительные системы третьего поколения имеют стандартный набор команд, который включает в себя 144 инструкции. Полный набор команд обеспечивает работу с фиксированной и плавающей запятой, десятичной арифметикой (выполнение арифметических операций над числами в десятичной системе счисления), алфавитно-цифровой информацией, с логическими операциями и операциями условных переходов. Таким образом, стандартный набор содержит команды, обслуживающие запросы по обработке данных для различных областей применения. Однако различные модели могут снабжаться любым требуемым набором команд из стандартного набора. Например, модель может не иметь десятичной арифметики или возможности работать с плавающей запятой. Но по мере необходимости требуемая возможность может быть добавлена. Это достигается за счет того, что большинство команд микропрограммированы.

Для увеличения производительности системы несколько моделей могут быть объединены в единый комплекс. Связь между отдельными центральными процессорами можно осуществить на трех уровнях:

1) использование общего быстродействующего устройства ввода-вывода, например дисков;

2) использование непосредственной связи между каналами двух отдельных систем;

3) использование общей основной памяти, что позволяет производить обмен информации со скоростью работы процессора.

Таким образом, отдельные модели системы получаются при таком выборе компонент системы, когда наиболее полно удовлетворяются многочисленные противоречивые требования к внутренней

производительности, функциональным возможностям и устройствам ввода-вывода.

Работа вычислительных систем осуществляется под управлением СУПЕРВИЗОРа. СУПЕРВИЗОР — это программа, которая координирует и выполняет все команды ввода-вывода, обрабатывает прерывания и особые случаи, осуществляет планирование и выполнение работ, обеспечивает одновременное выполнение нескольких программ. С точки зрения программиста, пользователя системы, СУПЕРВИЗОР и оборудование представляют собой единое целое.

Вычислительные системы работают в мультипрограммном режиме, т. е. одновременно могут выполняться несколько программ, принадлежащих одной или разным задачам. Это требует наличия средств защиты памяти, позволяющих запретить одной программе прочитать или записать информацию в область памяти, отведенную другой программе. Такие средства предназначены для защиты памяти в двух случаях: во время выполнения команд центральным процессором и во время выполнения операций ввода-вывода. Для защиты памяти во время выполнения команд основная память разделяется на блоки определенной длины (например, 2048 байт). С каждым блоком связывается четырехбитовый ключ памяти. Ключ не является частью основной памяти и поэтому недоступен пользователю. Для его установки и проверки существуют специальные макрокоманды СУПЕРВИЗОРа. Контроль защиты памяти осуществляет СУПЕРВИЗОР. При этом потери производительности машины не происходит.

Защита памяти во время выполнения операций ввода-вывода осуществляется с помощью проверки ключа данных программами каналов. Эти программы также являются составной частью СУПЕРВИЗОРа.

Кроме стандартного набора команд, вычислительная система может выполнять дополнительный ряд команд. Среди них можно выделить привилегированные и непривилегированные команды. Привилегированные команды выполняются только СУПЕРВИЗОРом и не доступны проблемным программам пользователя. Их появление в проблемных программах считается ошибкой. Непривилегированные команды доступны как СУПЕРВИЗОРу, так и проблемным программам. Под проблемными программами будем понимать программы, написанные пользователем. Примером привилегированных команд являются команды защиты памяти.

Вычислительные системы имеют также средства контроля правильности выполнения команд и обработки данных. Это позволяет отделить сбои машины от ошибок программы. При появлении ошибки возникает прерывание центрального процессора. Каждый тип ошибок обрабатывается специальной программой СУПЕРВИЗОРа. Для сокращения времени, необходимого для устранения неисправности при сбое машины, в заранее определенную ячейку памяти автоматически производится запись информации, необходимой для идентификации ошибки.

Для уменьшения числа случайных ошибок оператора пульты управления считаются устройствами ввода-вывода, и работа на этих пультах осуществляется под управлением одной из программ СУПЕРВИЗОРа. В дальнейшем пульт оператора будем называть консолью.

2.2. Структура системы

Логическая структура отдельных моделей вычислительных систем третьего поколения, представленная на рис. 2.1, включает центральный процессор, основную память, каналы, внешние устройства. Физическая реализация основной памяти, центрального процессора и каналов может быть различной для разных моделей вычислительной системы.

Центральный процессор предназначен для адресации основной и скоростной памяти, выборки и записи информации, арифметической и логической обработки данных, обеспечения нужного порядка следования команд, организации обмена между основной памятью и внешними устройствами. Логическая структура центрального процессора представлена на рис. 2.2. Объединяя несколько центральных процессоров, можно получить системы разной производительности. Диапазон таков, что отношение производительностей наиболее мощной модели к наименее мощной приблизительно равно 50 для научных расчетов и 15 для экономических.

Центральный процессор состоит из блока управления основной и скоростной памяти. Блок управления предназначен для организации работы системы.

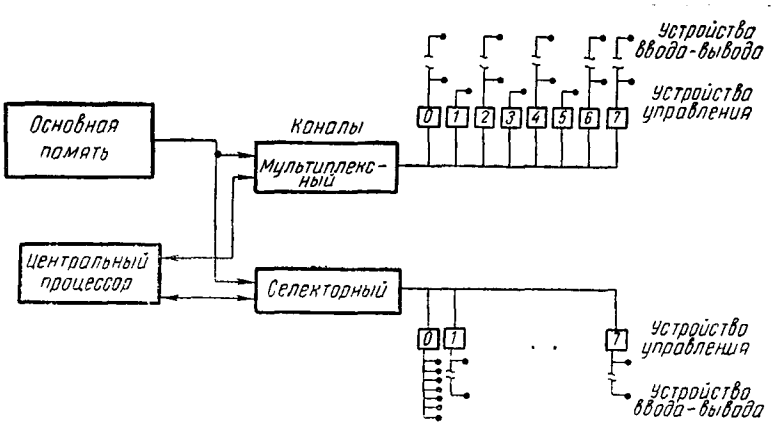


Рис. 2.1. Логическая структура модели

Основная память предназначена для хранения программ и данных. Она может наращиваться путем объединения быстродействую-

щих накопителей средних размеров и накопителей большой емкости со средним быстродействием. Использование иерархической организации блоков памяти в системе позволяет удовлетворять требования как производительности, так и большого объема памяти.

Различные модели системы отличаются друг от друга быстродействием памяти, количеством разрядов в регистрах и возможностью параллельной обработки информации. Тем не менее эти различия не ощущаются программистом с точки зрения использования памяти.

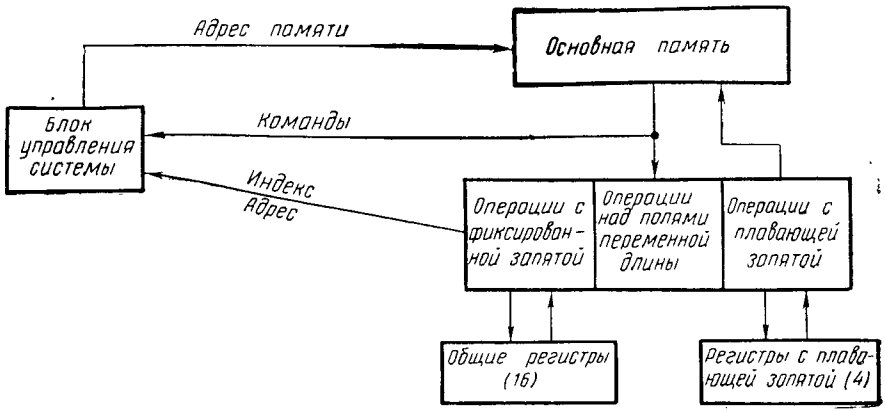


Рис. 2.2. Логическая структура процессора

Единица памяти, используемая при обмене между основной памятью (ОП) и центральным процессором (ЦП), называется *байтом*. Байт служит для построения любых форматов, используемых в системе. Емкость ОП определяется числом байтов. Байт содержит 8 информационных бит и один контрольный. Контрольный бит служит для обнаружения ошибок по четности и не может быть изменен программным путем.

Байты могут обрабатываться отдельно или объединенными группами. Группу последовательных байтов будем называть *полем*. Поле из двух байтов называется *полусловом*, из четырех — *словом*, из восьми — *двойным словом*.

Длина поля может определяться в команде неявно или явно. При неявном задании длина поля определяется кодом операции, при явном указывается специальными разрядами команды. Явное задание длины поля используется, когда оно имеет переменную длину, неявное — для полей фиксированной длины (полуслово, слово, двойное слово). В любом поле биты нумеруются слева направо, начиная с нуля.

Каждый байт основной памяти имеет свой адрес, который состоит из 24 битов. Адресация байтов начинается с нуля. За последним байтом основной памяти следует байт с адресом «нуль» вследствие того, что адресация организуется циклически. Поэтому, если в программе произойдет обращение к байту, адрес которого больше адреса последнего байта основной памяти, аппаратурно бу-

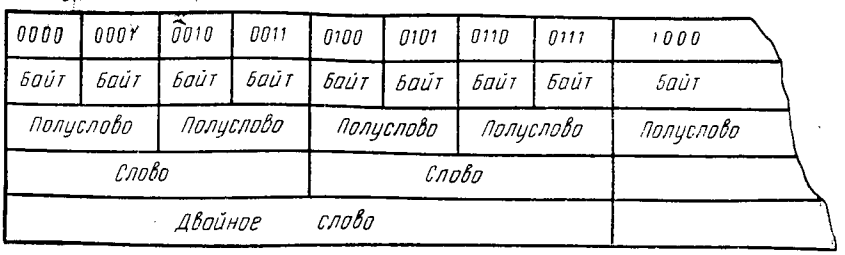


Рис. 2.3. Форматы полей

дет выдан сигнал об ошибке — неправильная адресация. Местонахождение любого поля в памяти определяется адресом его самого левого байта.

Поля фиксированной длины должны размещаться в памяти с *выравненного адреса.* Адрес называется выравненным, если его номер кратен числу байтов поля с фиксированной длиной. Например, слово должно располагаться в памяти с адреса, кратного четырем. Поля переменной длины могут начинаться с любого адреса. Действия, связанные с вычислением выравненного адреса, называются *выравниванием.*

Адреса записываются в двоичном коде. В компактном виде адрес записывается числом в шестнадцатиричной системе счисления. Например, расположение полей фиксированной длины в памяти представлено на рис. 2.3. Предположим, что десятичный адрес равен 104. Тогда в шестнадцатиричной системе счисления адрес поля будет равен 68, в двоичной — 000000000000000001101000. Адрес следующего двойного слова будет равен 70 в шестнадцатиричной системе счисления, 112 — в десятичной. При использовании шестнадцатиричной системы счисления *A* обозначает 10, *B* — 11, *C* — 12, *D* — 13, *E* — 14, *F* — 15.

Объем основной памяти для различных моделей колеблется в пределах от 8 *K* байт до 1024 *K* байт. *K* равно 1024 байтам.

Кроме основной памяти, имеется скоростная память, содержащая от 72 до 128 слов (по 32 бита). Эта память обладает более коротким циклом обращения, чем основная память, и доступна процессору с помощью регистров.

Центральный процессор имеет 16 общих регистров для операндов с фиксированной запятой и 4 регистра для операндов с плавающей запятой. В различных моделях физическая реализация регистров может быть различной.

Общие регистры могут быть использованы в качестве индексных регистров в операциях над адресами и при индексации, а также при хранении данных в арифметических и логических операциях с фиксированной запятой. В регистр можно поместить одно слово. Общие регистры пронумерованы от 0 до 15. Адрес общего регистра занимает поле в полубайт. Это поле в команде обозначается через *R*.

Для выполнения некоторых команд два смежных регистра могут использоваться совместно, позволяя работать с двойными словами. В этом случае адресуемый регистр содержит старшие разряды операнда и должен иметь четный адрес. Дополнительный нечетный регистр предназначен для хранения младших разрядов операнда.

Регистры плавающей запятой перенумерованы 0, 2, 4, 6 и имеют длину в два слова. В них могут размещаться как короткие (длиной в слово), так и длинные (длиной в два слова) операнды. Короткие операнды располагаются в старших разрядах регистра. Младшие разряды в этом случае игнорируются. Тип регистра, используемого в команде, определяется кодом операции.

Центральный процессор может выполнять операции следующих типов: операции с фиксированной запятой, операции над десятичными числами, операции с плавающей запятой и логические операции, условные переходы, операции над символами и др.

Операции различных типов задаются кодом операции и отличаются форматами используемых данных, применяемыми регистрами, способом задания длины поля.

Основной арифметический операнд представляет собой поле длиной в одно слово. Представление информации — с фиксированной запятой. Для увеличения производительности или для улучшения загрузки памяти операнды могут задаваться длиной в полуслово. Для сохранения точности некоторые произведения и все делимые имеют длину в двойное слово. Операции можно производить над целыми числами и над адресами. Такое комбинированное использование операций дает экономию средств и позволяет не делать различия при вычислениях, производим ли мы действия над адресами или над числами.

Операции над десятичными числами предусмотрены для тех видов обработки, где в промежутке между вводом и выводом данных выполняется небольшой объем вычислений.

Десятичные цифры выражаются в двоично-кодированной десятичной форме четырехбитовыми кодами от 0000 до 1001 для цифр 0—9 соответственно. Кодовые комбинации — 1011—1101 — служат для обозначения минуса, остальные кодовые комбинации — 1010, 1100, 1110, 1111 — служат для обозначения плюса.

Десятичные числа могут храниться в упакованном или зонированном формате. В упакованном формате в одном байте размещены две цифры. Знак числа помещается в крайнем правом полубайте (рис. 2.4а).

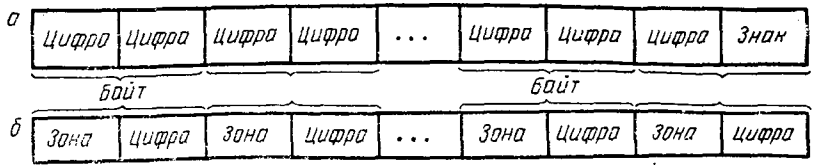


Рис. 2.4. Форматы представления десятичных чисел

Числа, представленные в зонированном формате (рис. 2.4б), являются подмножеством буквенно-числовых символов. В одном байте размещается одна цифра. Левые полбайта занимает зона, представляющая собой код 1111. Знак числа располагается вместо зоны в крайнем правом байте с последней цифрой числа. Зонированный формат используется в основном в операциях ввода-вывода. Примеры представления числа даны на рис. 2.5.

Числа с плавающей запятой могут располагаться в двух форматах фиксированной длины: коротком и длинном. Эти форматы отличаются только длиной мантиссы и представлены на рис. 2.6.

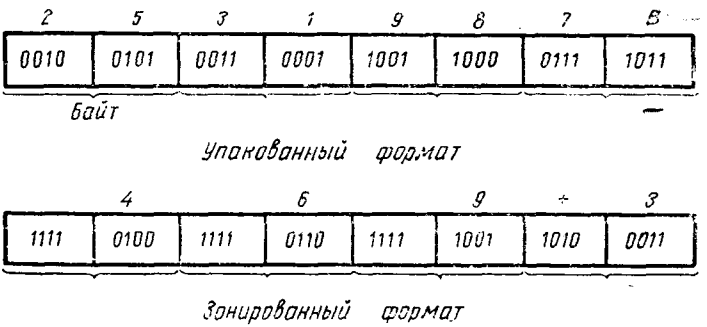


Рис. 2.5. Пример представления чисел

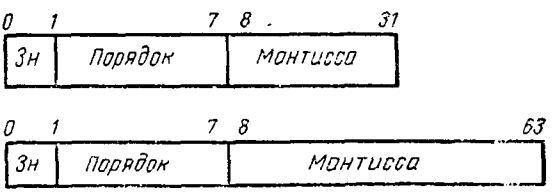


Рис. 2.6. Форматы чисел с плавающей запятой

Короткое число с плавающей запятой представляет собой поле длиной в одно слово. Длинное число — поле длиной в два слова. Знак числа расположен в нулевом бите (самом левом), порядок занимают биты 1—7, в первом бите находится знак порядка, мантисса занимает оставшиеся биты.

Логические операции выполняются над буквенно-числовой информацией фиксированной и переменной длины. Один символ занимает поле в один байт.

Рассмотрим форматы команд, выполняемых вычислительными системами.

Длина команды в зависимости от формата может быть равной одному, двум или трем полусловам (рис. 2.7). Команды длиной в полуслово оперируют с регистрами и не позволяют обращаться к основной памяти. Команды длиной в слово оперируют с одним адресом памяти. Команды длиной в три полуслова указывают два адреса памяти.

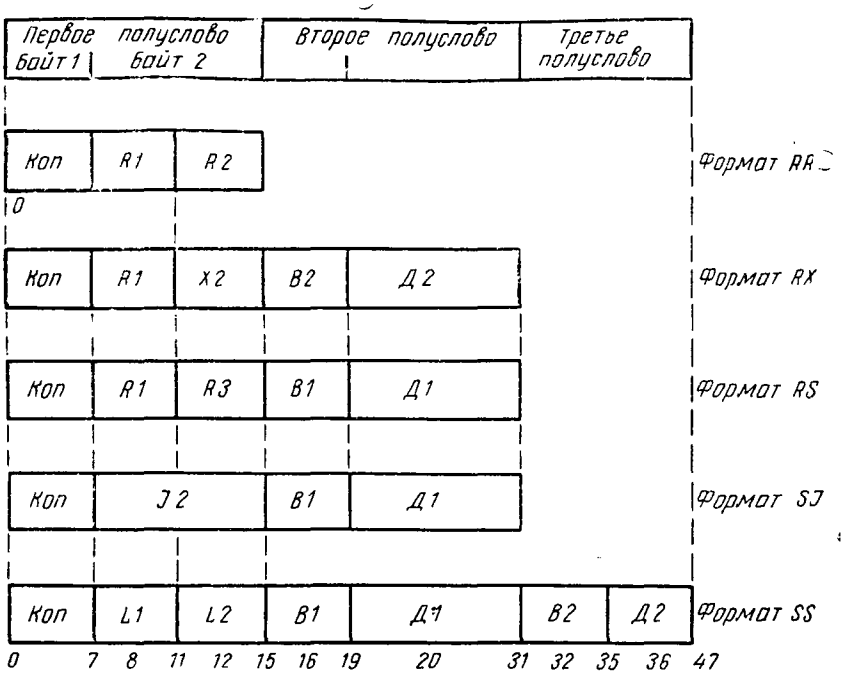


Рис. 2.7. Форматы команд

Команды имеют пять основных форматов *RR*, *RX*, *RS*, *SI*, *SS*. Формат *RR* обозначает операцию типа регистр — регистр; формат *RX* — операцию типа регистр — память, при этом адрес памяти индексируется; формат *RS* — операцию типа регистр — память без индексации. Формат *SI* обозначает операцию, один операнд которой находится в памяти, другой указан в самой команде. Формат *SS* определяет операцию типа память — память.

Абсолютный адрес обращения к основной памяти формируется из трех двоичных компонент. *Базовый адрес* представляет собой

24-битовое число, находящееся в общем регистре, номер которого указывается в команде полем *B*.

Индекс представляет собой 24-битовое число, находящееся в общем регистре, номер которого указывается в команде полем *X*.

Смещение представляет собой 12-битовое число, являющееся смещением относительно базового адреса. Смещение задается в команде полем *D*.

Таким образом, абсолютный адрес *E* будет равен сумме содержимого общего регистра *B*, хранящего базис, общего регистра *X*, содержащего индекс, и смещения адреса от начала базиса:

$$E = (B) + (X) + D.$$

Вычислительные системы третьего поколения используют стандартные способы для подключения устройств ввода-вывода, которые могут отличаться друг от друга по обозначению, скорости передачи данных, времени доступа.

Связь между центральным процессором и устройствами ввода-вывода осуществляется с помощью каналов. Каждый канал имеет свою собственную линию связи с центральным процессором и работает асинхронно по отношению к процессору. Различные модели могут иметь от двух до шестнадцати каналов.

Имеются два типа каналов: *мультиплексный* и *селекторный*. Оба типа каналов связаны с процессором посредством сопряжения с устройством управления. Связь с периферийными устройствами осуществляется подканалами.

Селекторный канал имеет один подканал и работает с отдельным устройством ввода-вывода до тех пор, пока не будет завершена передача всех данных по этому устройству. Этот канал используется для обслуживания быстродействующих устройств ввода-вывода, включающих устройства с прямым доступом на постоянных и сменных дисках, семи- и девятидорожковые магнитные ленты, магнитные барабаны, магнокарты.

Мультиплексный канал имеет до восьми подканалов и может работать либо с одним устройством, либо одновременно обслуживать несколько устройств. Этот канал используется для связи с медленными устройствами, включающими считывающие устройства с перфолент и перфокарт, перфораторы лент и карт, различные печатающие устройства, читающие устройства с бланков, телетайпы.

Каждое устройство ввода-вывода связано с одним или несколькими каналами через сопряжение. Сопряжение позволяет подсоединять как существующие устройства, ввода-вывода, так и те, которые будут разрабатываться для нужд заказчика без изменения системы команд или характера работы каналов. В тех случаях, когда необходимо согласовать внутренние связи устройства ввода-вывода с сопряжением, используется устройство управления.

С помощью селекторных и мультиплексных каналов можно подсоединить к процессору до 256 внешних устройств. Эти устройства можно адресовать с помощью чисел от 0 до 255. Адрес любого уст-

ройства состоит из номера канала, к которому оно подсоединено, и собственного номера.

Обмен информацией между основной памятью и устройствами ввода-вывода производится байтами через специальные регистры-буферы. Для осуществления обмена с любым устройством имеется четыре команды процессора, выполняемые СУПЕРВИЗОРом: НАЧАТЬ ВВОД-ВЫВОД, ОПРОСИТЬ КАНАЛ, ОПРОСИТЬ ВВОД-ВЫВОД, ОСТАНОВИТЬ ВВОД-ВЫВОД.

Команда НАЧАТЬ ВВОД-ВЫВОД предназначена для инициации операции ввода-вывода. Все операции ввода-вывода начинаются по этой команде. Команда ОСТАНОВИТЬ ВВОД-ВЫВОД прекращает выполнение любой операции ввода-вывода. Команда ОПРОСИТЬ КАНАЛ используется для получения информации, отражающей состояние адресуемого канала. По этой информации можно выяснить, например, доступен ли канал или занят выполнением операции, включен ли он, хранится ли в нем сигнал прерывания. Команда ОПРОСИТЬ ВВОД-ВЫВОД используется для получения информации о состоянии адресуемого канала, подканала и устройства ввода-вывода.

Имеются шесть команд канала: ЧИТАТЬ, ПИСАТЬ, ЧИТАТЬ В ОБРАТНОМ НАПРАВЛЕНИИ, УПРАВЛЕНИЕ, ПЕРЕХОД В КАНАЛ, УТОЧНИТЬ СОСТОЯНИЕ.

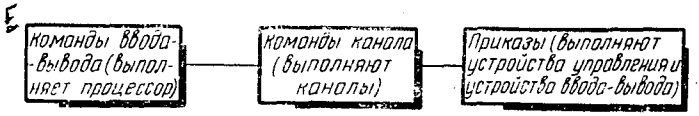


Рис. 2.8. Взаимосвязь команд ввода-вывода

По команде ЧИТАТЬ производится чтение с устройства в заданную область основной памяти. По команде ПИСАТЬ производится запись в указанное устройство из области основной памяти. По команде ЧИТАТЬ В ОБРАТНОМ НАПРАВЛЕНИИ производится чтение с внешнего носителя, который двигается в обратном направлении. По команде УПРАВЛЕНИЕ производится перемещение управляющих слов в устройстве для определения различных действий устройства (типа: поиск отдельной дорожки на диске, продвижение бумаги на устройстве печати). Эти управляющие слова называются приказами. Каждое устройство ввода-вывода имеет характерный для него набор приказов.

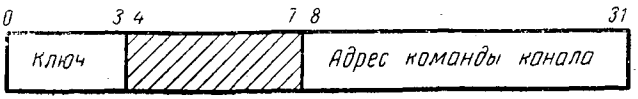


Рис. 2.9. Структура адресного слова канала

(в битах 0—3 находится ключ защиты области памяти, с которой производится обмен; в битах 8—31 находится адрес слова команды канала; биты 4—7 резервируются)

Команда ПЕРЕХОД В КАНАЛЕ используется, чтобы нарушить последовательность выполнения команд канала и передать управление на другую цепочку команд канала. Цепочка — последовательность команд канала для выяснения какого-либо действия.

По команде УТОЧНИТЬ СОСТОЯНИЕ в заданную область основной памяти передается информация, характеризующая состояние устройства ввода-вывода. Эта информация имеет смысл только для заданного устройства.

Таким образом, для выполнения операции ввода-вывода используются команды ввода-вывода процессора, команды канала и приказы. Взаимосвязь между ними приведена на рис. 2.8. Операции ввода-вывода выполняются только СУПЕРВИЗОРом. Выполнение операций ввода-вывода осуществляется в следующем порядке.

По команде процессора НАЧАТЬ ВВОД-ВЫВОД инициируются действия канала. Информация адресного слова канала (АСК) из постоянно фиксированной области памяти пересылается в управляющие регистры канала. Формат адресного слова канала приведен на рис. 2.9. Действие команды заканчивается, и дальнейшая работа выполняется каналом независимо и параллельно с действиями процессора.

Из адресного слова канала выбирается адрес слова команды канала (СКК), представляющий собой начальный адрес программ обмена с данным устройством. Структура СКК приведена на рис. 2.10.

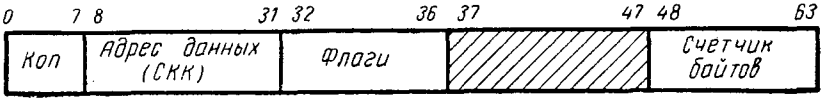


Рис. 2.10. Структура слова команды канала:

Биты 0—7 содержат код операции канала; 8—31 — адрес байта основной памяти, с которого начинается обмен, или адрес следующей команды канала; 32 — флаг, указывающий, что адрес данных дается в следующей команде канала; 33 — флаг, указывающий, что нужно выполнить следующую команду канала; 34—35 — флаги сбойных прерываний; 36 — флаг программно-управляемого прерывания; 37—47 — резервируется; 48—63 — счетчик числа байтов, участвующих в операции

После окончания перемещения данных от устройства или к устройству проверяется условие окончания операции. Если перемещение данных было закончено нормально, каналом вырабатывается флаг прерывания в регистре флагов прерывания по окончании обмена, и управление передается процессору. При этом вся информация о ходе обмена фиксируется в слове состояния канала (ССК), структура которого приведена на рис. 2.11. Слово состояния канала хранится в постоянно фиксируемой области основной памяти.

Если перемещение данных было закончено аварийно, то вырабатываются соответствующие флаги сбойных ситуаций, которые также размещаются в слове состояния канала, и управление передается процессору.

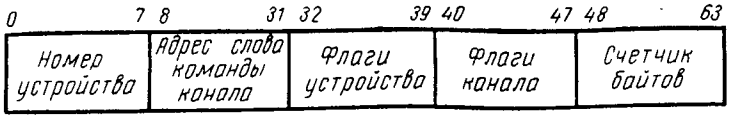


Рис. 2.11. Структура слова состояния канала

Флаги устройства и флаги канала содержат необходимую характеристику о ходе обмена (занято устройство/канал или нет, закончена ли операция обмена и т. д.). Счетчик байтов содержит текущее количество обмениваемых байтов.

Для уточнения прохождения операции ввода-вывода и выявления состояния устройства используются команды ОПРОСИТЬ ВВОД-ВЫВОД и ОПРОСИТЬ КАНАЛ. В случае нормального окончания операции выполняется команда ОСТАНОВИТЬ ВВОД-ВЫВОД. В случае аварийного окончания должны быть приняты соответствующие меры и также выполнена команда ОСТАНОВИТЬ ВВОД-ВЫВОД. Все эти действия выполняются специальными программами СУПЕРВИЗОРа.

2.3. Система прерываний

Обычное выполнение программы представляет собой последовательную выборку команд. Нарушение последовательности операций может происходить при переходе, переключении состояния процессора, прерывании от устройств ввода-вывода или при вмешательстве оператора. Последние три типа передачи управления изменяют не только адрес команды, но и всю существенную информацию о состоянии программы и машины. Они образуют развитую сеть прерываний. Прерывание позволяет автоматически изменять состояние процессора.

Все прерывания можно разделить на пять классов: программные прерывания, вызываемые ошибками в программах или использованием недопустимых операций; внешние прерывания от нажатия кнопки на пульте управления системы, от часов, от сигналов, предназначенных для прямого управления системой; прерывания для обращения к супервизору, когда требуются действия операционной системы для нормального прохождения вычислений; прерывания от ввода-вывода, позволяющие процессору отвечать на сигналы, поступающие от каналов и устройств ввода-вывода; прерывания от схем контроля машины при сбоях.

Прерывание возникает при появлении соответствующего условия как внутри самого процессора, так и от устройств ввода-вывода. Всего имеется 32 условия прерывания. Каждое условие имеет строго определенный приоритет обслуживания. Прерывание с высшим приоритетом обслуживается первым.

Информация, необходимая для обработки прерываний, запоминается в специальных регистрах. Будем считать, что такими регистрами являются регистр прерываний, регистр состояния прерывания, регистр маски прерывания и программный счетчик.

На основании информации, заключенной в этих регистрах, осуществляются управление порядком выборки команд, фиксация и индикация состояния процессора по отношению к выполняемой программе.

Информацию, характеризующую программу в любой момент времени, назовем состоянием программы. Состояние программы также определяется содержимым регистров. Состояние программы, используемое в данной момент времени, называется текущим. Запоминая текущее состояние программы во время прерывания, можно сохранить информацию о ней. Эта информация используется для последующего анализа и для восстановления нормального продолжения выполнения прерванной программы после обработки прерывания.

Таблица 2.1

Адрес основной памяти	Длина	Назначение
00	Двойное слово	Используются программой начальной загрузки
08	То же	
16	»	
24	»	ССП для внешних прерываний
32	»	ССП для обращения к супервизору
40	»	ССП для программных прерываний
48	»	ССП для прерываний от схем контроля машины
56	»	ССП для прерываний от ввода-вывода
64	»	Слово состояния канала
72	Слово	Адрес слова команды канала
76	»	Резервируется
80	»	Таймер
84	»	Адрес таблицы адресов
88	Двойное слово	НСП для внешних прерываний
96	То же	НСП для обращения к супервизору
104	»	НСП для программных прерываний
112	»	НСП для прерываний от схем контроля машины
120	»	НСП для прерываний от ввода-вывода
128	»	Область памяти для информации о состоянии машины при диагностике (зависит от конфигурации)

Каждому классу прерываний соответствуют два состояния программы: «старое» и «новое», которые размещаются в постоянно фиксированной области основной памяти (см. табл. 2.1). При возникновении прерывания текущее состояние программы запоминается в соответствующей данному классу прерываний области основной памяти — старое состояние программы (ССП). Затем производится выборка нового состояния программы (НСП), которое становится текущим.

Каждое прерывание может быть запрещено или разрешено посредством задания маски прерывания, хранимой в регистре маски прерывания для каждого состояния процессора. Если прерывание разрешено (т. е. незамаскировано), то автоматически запоминается текущее состояние процессора и различных индикаторов для формирования состояния программы. Если прерывание запрещено (т. е. замаскировано), то управление передается на выполнение следующей команды программы. Как только прерывание поступает на обработку, происходит восстановление соответствующего бита в регистре прерываний.

Прерывание производится всегда после того, как выполнение текущей команды программы закончилось, а выполнение следующей не началось. Однако выполнение текущей команды может быть нарушено. Чтобы обеспечить правильную работу процессора после прерывания, запоминается причина прерывания.

Во время выполнения команды одновременно может поступить несколько запросов на прерывание. Они устанавливаются в очередь в регистре прерываний в соответствии со своим приоритетом. Приоритет определяется весом прерывания, который устанавливается аппаратурно. Порядок рассмотрения прерываний строго определен: прерывания от схем контроля машины, которые блокируют все другие прерывания; программные прерывания или прерывания при обращении к СУПЕРВИЗОРу (они не могут появиться одновременно); внешние прерывания; прерывания от ввода-вывода.

Необходимо выделить прерывания по обращению к СУПЕРВИЗОРу. Прерывание может произойти как аппаратурно (переключение состояний процессора), так и при выполнении непривилегированной команды вызова СУПЕРВИЗОРа. Мнемоническое обозначение этой команды — *SVC*. При вызове СУПЕРВИЗОРа с помощью команды обращения к нему код прерывания берется из младших восьми бит регистра состояния прерывания. При установленном бите в регистре прерываний в зависимости от маски, находящейся в регистре масок прерываний, происходит или нет прерывание процессора для передачи управления СУПЕРВИЗОРу.

Из внешних прерываний выделим прерывание, поступающее от таймера. Таймер служит для организации программным способом часов, показывающих время суток. Таймер представляет собой слово, хранящееся в основной памяти с адреса 80 (см. табл. 2.1). Из этого слова производится вычитание с частотой 50 или 60 гц в зависимости от частоты электросети. Содержимое таймера рассматри-

вается как целое число со знаком и может обрабатываться по правилам операций с фиксированной запятой. Когда значение таймера переходит из положительного в отрицательное, выдается сигнал внешнего прерывания. Полный цикл таймера составляет 15,5 часа. Значение таймера доступно в конце выполнения любой команды. В качестве датчика времени таймер используется для измерения относительно небольших интервалов.

2.4. Состояние центрального процессора

Центральный процессор может функционировать в одном из четырех независимых состояний: нормальная обработка — счет (состояние Р1), обработка прерывания (состояние Р2), анализ прерываний (состояние Р3), прерывание от схем контроля машины (состояние Р4).

Состояние Р1 представляет собой обычное выполнение проблемных программ. Состояние Р2 используется СУПЕРВИЗОРом для выполнения действий, требуемых каждым прерыванием.

Переключение в состояние Р3 происходит всегда автоматически при возникновении прерывания, кроме прерываний от схем контроля машины. В этом состоянии СУПЕРВИЗОР производит анализ типа прерывания и определяет требуемые действия. При необходимости осуществляется связь с программами состояния Р2.

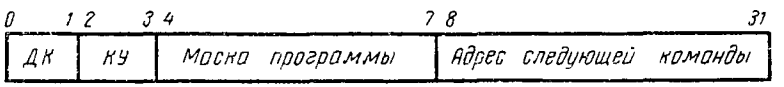


Рис. 2.12. Структура программного счетчика:

ДК — код длины выполняемой команды; 01 — длина команды — полуслова; ДК—10 — длина команды — слова; 11 — длина команды — три полуслова; КУ — код условия прерывания

Переход в состояние Р4 осуществляется автоматически при появлении прерываний от схем контроля машины.

Для каждого из четырех состояний отводится несколько регистров скоростной памяти и собственный регистр управления. Такими регистрами являются программный счетчик (Р-счетчик), общие регистры, регистры с плавающей запятой, регистр состояния прерывания, регистр маски прерывания и регистр прерывания. Общие регистры и регистры с плавающей запятой рассматривались в 2.2. Программный счетчик является регистром скоростной памяти. Его формат приведен на рис. 2.12. Он служит для определения адреса следующей команды, которой надо будет передать управление, а также содержит характеристику выполняемой команды.

Код условия прерывания устанавливается аппаратурно в момент появления прерывания, а затем переносится в регистр состояния пре-

рывания. Маска программы содержит флаги, разрешающие или запрещающие обработку некоторых программных ситуаций, например переполнение с фиксированной запятой, переполнение десятичной арифметики.

Регистр состояний прерывания содержит информацию о прерывании. Его структура представлена на рис. 2.13.

Индикатор паритета основной памяти устанавливается в единицу для состояния процессора Р4 при обнаружении ошибок паритета основной памяти. Индикатор ошибки устанавливается в единицу для состояния процессора Р4 при обнаружении ошибки паритета скоростной памяти. Ключ защиты предназначен для сравнения с ключом защиты основной памяти во всех операциях записи в память или чтения.

Для представления десятичной информации используется несколько способов кодирования, где *К* определяет тип кодирования.

Признак *П* устанавливается в единицу для состояния процессора Р1, если команда не является привилегированной, и в нуль — для программ СУПЕРВИЗОРа. Для других состояний процессора признак *П* устанавливается в нуль, что означает использование привилегированных команд.

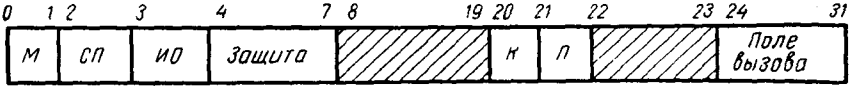


Рис. 2.13. Структура регистра состояния прерывания:

М — индикатор паритета основной памяти; СП — состояние процессора, при котором произошло прерывание;

СП = { 00 — Р4; 01 — Р3; 10 — Р2; 11 — Р1

ИО — индикатор ошибки паритета скоростей памяти; К — код представления десятичной информации; П — признак разрешения использовать привилегированные команды

Поле вызова предназначено для запоминания номеров общих регистров, используемых в прерванной команде.

Регистр маски прерываний содержит 32 бита и находится в скоростной памяти. Каждый бит связан с условием прерывания. Установление какого-либо бита в 0 запрещает прерывание по этому условию. Установление в 1 разрешает прерывание при появлении соответствующего условия прерывания.

Регистр прерываний также расположен в скоростной памяти и содержит 32 бита, каждый из которых связан с условием прерывания. Содержимое регистра доступно программе, чтобы посмотреть, какие ожидаются прерывания.

Таким образом, состояние процессора характеризуется единственным для всех состояний регистром прерываний, четырьмя регист-

рами маски прерываний (по одному на каждое состояние), четырьмя программными счетчиками (по одному на каждое состояние), любым количеством общих регистров (из 16 имеющихся) и регистров с плавающей запятой (последние используются только в состоянии P1).

Изменение состояния может произойти программно или автоматически. В любом случае адресация к регистрам может быть произведена обычным путем с использованием общих регистров 0—15. Доступ к регистрам, находящимся в скоростной памяти, осуществляется с помощью привилегированных команд ЗАГРУЗИТЬ СКОРОСТНУЮ ПАМЯТЬ и ХРАНИТЬ СКОРОСТНУЮ ПАМЯТЬ, которые определяют абсолютные адреса требуемых регистров.

ВОПРОСЫ К ГЛАВЕ

1. Какими свойствами обладают вычислительные системы третьего поколения?
2. Что означает совместимость «вверх» и «вниз»?
3. Как организуется защита памяти и зачем она нужна?
4. Назовите основные составляющие вычислительной системы.
5. Что является минимальной единицей основной памяти?
6. Какие поля вы знаете?
7. Назовите форматы команд и их назначение.
8. Назовите типы каналов и их назначение.
9. В какой последовательности осуществляется операция ввода-вывода?
10. Назовите состояние процессора и его назначение.
11. Назовите классы прерываний, дайте их определение.
12. Для чего служит маска прерывания?

Глава 3

РЕЖИМЫ ЭКСПЛУАТАЦИИ СИСТЕМ

Бурный рост потребностей в высокопроизводительных и надежных средствах вычислений привел к появлению и развитию вычислительных систем (ВС). Пока рано говорить о достаточно полной и точной классификации ВС, так как развитие их структур и способов функционирования продолжается, и почти каждая новая разработка ВС вносит новые идеи, заставляющие изменять принятую классификацию. Тем не менее можно указать некоторые уже установившиеся признаки, по которым удается классифицировать существующие и проектируемые ВС (рис. 3.1).

В приведенной классификации не нашли отражения способы использования ВС при решении задач и организации вычислительного процесса в системе машин. Первый опыт эксплуатации ВС показал, что большие возможности этих систем могут быть полностью раскрыты и использованы только при применении эффективных методов и средств организации работы. Обзор литературы по применению вычислительной техники позволяет обобщить различные формы эксплуатации ВС и представить их в виде определенной классифи-

кационной схемы (рис. 3.2). При этом следует заметить, что в настоящее время наибольшее внимание в литературе уделяется организации работ с разделением времени.

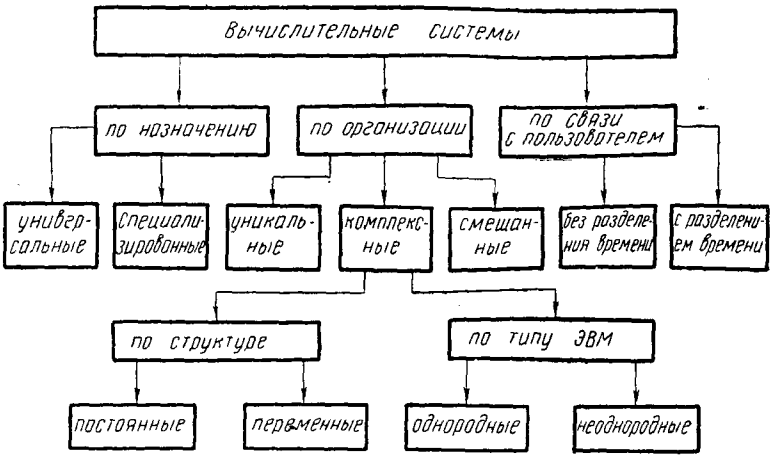


Рис. 3.1. Классификационная схема ВС

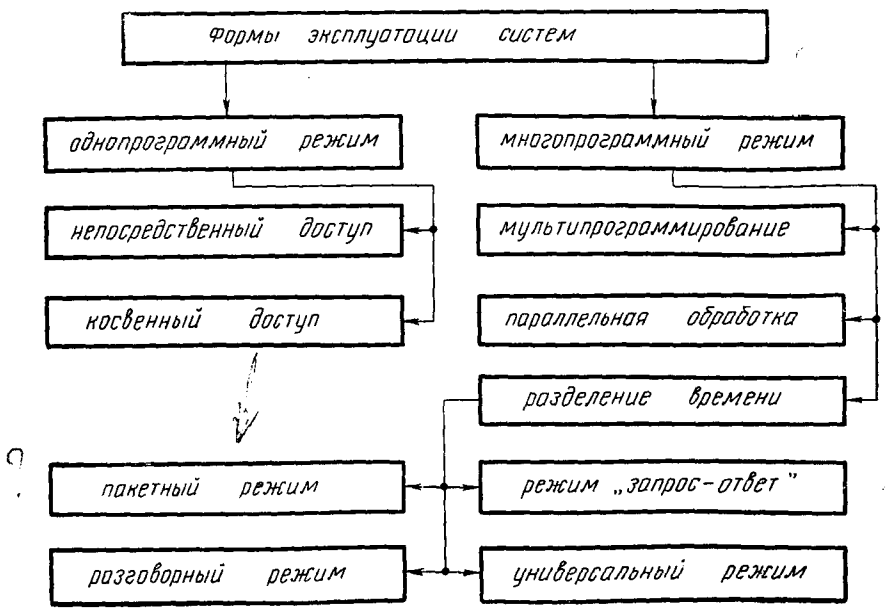


Рис. 3.2. Классификационная схема форм эксплуатации систем

3.1. Однопрограммный режим работы

Вычислительная система (машина) работает в однопрограммном режиме, если один или несколько процессоров, входящих в ее состав, обслуживают одного пользователя.

В начальной стадии применения ЭВМ пользователь сам работал за пультом управления, осуществляя ввод своей программы, ее запуск и наблюдал за выводом результатов по мере их получения.

Время реакции пользователя при непосредственном доступе велико по сравнению со временем реакции машины, что чрезвычайно невыгодно с точки зрения производительности последней. Действительно, пользователь определенное время загружает колоды перфокарт или перфолент, набирает с пульта управляющую информацию, наконец, просто обдумывает свое очередное решение, в это время ЭВМ простаивает. При пакетной обработке доступ к ЭВМ осуществлялся косвенно посредством управляющей программы, обеспечивающей переход от задачи к задаче и контроль за их выполнением, значительно сокращались непроизводительные потери машинного времени, повышалась эффективность использования машин, но в то же время полностью исключалась возможность диалога человек — машина.

При пакетном режиме предусматривается последовательная обработка задач. В некоторый момент времени только одна из задач пакета выполняется, а остальные задачи (программы) пакета находятся в состоянии ожидания своей очереди. Во время выполнения одной из программ пакета запрещается ее прерывание с целью перехода к другой программе пакета. Переход к очередной программе разрешается либо после завершения текущей программы, либо в случае ее аварийного останова.

Для внесения малейшего изменения в программу необходимо ожидать ввода в машину очередного пакета задач. Таким образом, при косвенном доступе к ЭВМ время реакции машины в большинстве случаев превышает время реакции пользователя.

Математическая модель однопрограммного режима эксплуатации ЭВМ

С целью определения расчетных характеристик однопрограммного режима эксплуатации ЭВМ построим математическую модель этого процесса. Для математического описания процесса функционирования однопрограммного режима эксплуатации ЭВМ представим ее в виде системы массового обслуживания (рис. 3.3).

Анализ рассматриваемой модели проводится при следующих предположениях:

1) поток заявок (задач) от нескольких пользователей аппроксимируется пуассоновским потоком с интенсивностью λ, и заявки не покидают систему, пока не обслужатся;

2) обслуживающий прибор выходит время от времени из строя по закону Пуассона с параметром λ_0. За время восстановления прибора новые заявки продолжают поступать и становятся в очередь на обслуживание, т. е. имеет место система без потерь. Предполагаем, что выход прибора из рабочего состояния возможен как в период обслуживания, так и в период, когда он не занят обслуживанием;

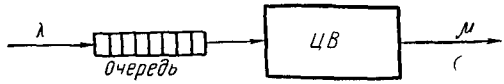


Рис. 3.3. Схема однопрограммного режима

3) считаем, что время обслуживания одной заявки и время восстановления прибора подчиняются показательному закону с параметрами соответственно μ и μ_0;

4) заявки обслуживаются по принципу «первой пришла — первая обслужена».

К основным характеристикам функционирования такой системы относятся:

математическое ожидание и дисперсия числа заявок в системе;

математическое ожидание длины очереди;

среднее время пребывания заявки в системе и ожидания в очереди;

вероятность того, что в системе находится K заявок при исправном приборе и во время восстановления прибора.

Предположим, что процесс перехода из одного состояния в другие является марковским. Получаем следующую систему дифференциальных уравнений:

$$\left.\begin{aligned} P_0'(t) &= -(\lambda_0+\lambda)P_0(t)+\mu P_1(t)+\mu_0 H_0(t)\\ P_K'(t) &= -(\lambda_0+\lambda+\mu)P_K(t)+\lambda P_{K-1}(t)+\mu P_{K+1}(t)+\\ &\quad +\mu_0 H_K(t)\\ H_0'(t) &= -(\lambda+\mu_0)H_0(t)+\lambda_0 P_0(t)\\ H_K'(t) &= -(\lambda+\mu_0)H_K(t)+\lambda_0 P_K(t)+\lambda_0 H_{K-1}(t) \end{aligned}\right\} \qquad (3.1)$$

где $P_0(t)$ — вероятность того, что в момент t в системе отсутствуют заявки и прибор исправен;

$P_K(t)$ — вероятность того, что в момент t в системе находится K заявок и прибор исправен;

$H_0(t)$ — вероятность того, что в момент t в системе отсутствуют заявки и происходит восстановление прибора;

$H_K(t)$ — вероятность того, что в период восстановления прибора в системе находится K заявок.

Система (3.1) имеет бесконечное число линейных однородных дифференциальных уравнений. Ограничимся исследованием установившегося режима и отысканием предельных значений характеристик системы обслуживания. Для стационарного процесса на основании системы (3.1) получаем следующую систему линейных алгебраических уравнений:

$$\left.\begin{aligned}(\lambda_0+\lambda)\,p_0&=\mu p_1+\mu_0 h_0\\(\lambda_0+\lambda+\mu)\,p_k&=\lambda p_{k-1}+\mu p_{k+1}+\mu_0 h_k\\(\lambda+\mu_0)\,h_0&=\lambda_0 p_0\\(\lambda+\mu_0)\,h_k&=\lambda_0 p_k+\lambda_0 h_{k-1}\end{aligned}\right\}. \tag{3.2}$$

Для получения характеристик системы воспользуемся методом производящих функций. Вероятность того, что система занята обслуживанием заявки, определим по формуле

$$p=\sum_{k=1}^{\infty} p_k. \tag{3.3}$$

Аналогично вероятность того, что система занята восстановлением прибора

$$h=\sum_{k=0}^{\infty} h_k \tag{3.4}$$

Так как система может находиться в одном из возможных состояний, т. е. занята обслуживанием заявки, восстановлением прибора или свободна от того и другого (p_0), сумма вероятностей этих состояний равна 1:

$$p+h+p_0=1. \tag{3.5}$$

Исходя из этого введем производящую функцию:

$$F(x)=F_1(x)+F_2(x)+p_0, \tag{3.6}$$

где

$$F_1(x)=\sum_{k=1}^{\infty} p_k x^k, \tag{3.7}$$

$$F_2(x)=\sum_{k=0}^{\infty} h_k x^k. \tag{3.8}$$

Функции $F_1(x)$ и $F_2(x)$ являются определенными функциями при $|x|\leqslant 1$. Заметим, что $F_1(1)=p$, $F_2(1)=h$ и $F(1)=1$. Производящую функцию находим следующим образом. На основании двух первых уравнений системы (3.2), умножив их на x^k и просуммировав по всем k, получим зависимость между частичными производящими функциями $F_1(x)$ и $F_2(x)$:

$$\left(\lambda_0+\lambda+\mu-\lambda x-\frac{\mu}{x}\right)F_1(x)=\mu_0 F_2(x)-(\lambda_0+\lambda-\lambda x)\,p_0. \tag{3.9}$$

Аналогично, умножив два последних уравнения системы (3.2) на x^k и просуммировав по всем k, получим новую зависимость между $F_1(x)$ и $F_2(x)$:

$$(\lambda + \mu_0 - \lambda x) F_2(x) = \lambda_0 F_1(x) + \lambda_0 p_0. \tag{3.10}$$

Решая совместно уравнения (3.9) и (3.10), находим выражения для частичных производящих функций:

$$F_1(x) = \frac{\lambda_0 + \lambda + \mu_0 - \lambda x}{(\lambda + \mu_0 - \lambda x)(\mu - \lambda x) - \lambda_0 \lambda x} p_0, \tag{3.11}$$

$$F_2(x) = \frac{\mu \lambda_0}{(\lambda + \mu_0 - \lambda x)(\mu - \lambda x) - \lambda_0 \lambda x} p_0. \tag{3.12}$$

Подставив значения $F_1(x)$ и $F_2(x)$ из (3.11) и (3.12) в (3.6), сведя подобные, получим производящую функцию:

$$F(x) = \mu \frac{\lambda_0 + \lambda + \mu_0 - \lambda x}{(\lambda + \mu_0 - \lambda x)(\mu - \lambda x) - \lambda_0 \lambda x} p_0. \tag{3.13}$$

Учитывая, что $F(1) = 1$, находим вероятность того, что в системе отсутствуют заявки и прибор исправен:

$$p_0 = \frac{1 - \rho(1 + \rho_0)}{1 + \rho_0}, \tag{3.14}$$

где $\rho = \frac{\lambda}{\mu}$ — коэффициент загрузки обслуживающего прибора заявками;

$\rho_0 = \frac{\lambda_0}{\mu_0}$ — коэффициент загрузки обслуживающего прибора отказами.

Из (3.11) при $x \longrightarrow 1$ и (3.14) находим вероятность того, что система занята обслуживанием требования:

$$p = F_1(1) = \rho. \tag{3.15}$$

Такое выражение для p можно было предположить, так как рассматривается система с ожиданием. Аналогично из (3.12) при $x \longrightarrow 1$ и (3.14) находим вероятность того, что система занята восстановлением прибора:

$$h = F_2(1) = \frac{\rho_0}{1 + \rho_0}. \tag{3.16}$$

Коэффициент готовности прибора к обслуживанию:

$$k_\text{г} = 1 - h = \frac{1}{1 + \rho_0}. \tag{3.17}$$

Среднее количество заявок в системе (M) получаем, взяв производную по x от производящей функции (3.13) при $x \longrightarrow 1$:

$$\left.\frac{dF(x)}{dx}\right|_{x=1} = \lambda\mu \frac{\lambda_0 \mu + (\lambda_0 + \mu_0)^2}{[\mu_0(\mu - \lambda) - \lambda\lambda_0]^2} \cdot p_0, \tag{3.18}$$

так как

$$\left.\frac{dF(x)}{dx}\right|_{x=1} = \sum_{k=1}^{\infty} k x^{k-1} (p_k + h_k)|_{x=1} = \sum_{k=1}^{\infty} k (p_k + h_k) = M.$$

Подставив значение p_0 в (3.18) и использовав соответствующие обозначения, найдем:

$$M = \frac{\rho}{1+\rho_0} \cdot \frac{\alpha\rho_0 + (1+\rho_0)^2}{1-\rho(1+\rho_0)}, \tag{3.19}$$

где $\alpha = \frac{\mu}{\mu_0}$

В связи с тем что среднее число заявок в системе определяется суммой среднего их числа в очереди L и на обслуживании p, т. е.

$$M = L + p, \tag{3.20}$$

то для средней длины очереди получаем следующее выражение

$$L = \rho\left[\frac{\alpha\rho_0 + (1+\rho_0)^2}{1+\rho_0-\rho(1+\rho_0)^2} - 1\right]. \tag{3.21}$$

Среднее время пребывания заявки в системе будет определяться по формуле

$$W = \frac{1}{\mu(1+\rho_0)} \cdot \frac{\alpha\rho_0 + (1+\rho_0)^2}{1-\rho(1+\rho_0)}, \tag{3.22}$$

а среднее время ожидания в очереди — по формуле

$$W' = \frac{1}{\mu}\left[\frac{\alpha\rho_0 + (1+\rho_0)^2}{1+\rho_0-\rho(1+\rho_0)^2} - 1\right]. \tag{3.23}$$

Дисперсию числа заявок в системе D находим из второй производной по x от производящей функции (3.13) при $x \longrightarrow 1$

$$\left.\frac{d^2F(x)}{dx^2}\right|_{x=1} = 2\lambda^2\mu \frac{\lambda_0\mu(\mu-\lambda) + 2\lambda_0\mu(\lambda_0+\mu_0) + (\lambda_0+\mu_0)^3}{[\mu_0(\mu-\lambda) - \lambda_0\lambda]^3} p_0, \tag{3.24}$$

так как

$$\left.\frac{d^2F(x)}{dx^2}\right|_{x=1} = \sum_{k=1}^{\infty} k(k-1)\, x^{k-2}(p_k + h_k)]_{x=1} = M_2 - M,$$

где M_2 — второй начальный момент, а дисперсия

$$D = M_2 - M^2. \tag{3.25}$$

Подставив в (3.24) значение p_0 и использовав соответствующие обозначения, найдем второй начальный момент:

$$M_2 = 2\rho^2 \frac{\alpha\rho_0[\alpha(1-\rho) + 2(1+\rho_0)] + (1+\rho_0)^2}{(1+\rho_0)[1-\rho(1+\rho_0)]^2} + M. \tag{3.26}$$

Подставив (3.19) и (3.26) в (3.25), найдем дисперсию:

$$D = \frac{\rho}{[1-\rho(1+\rho_0)]^2}\left[1 + \rho_0 + \alpha\rho_0\left(\rho + \frac{1}{1+\rho_0}\right) + \frac{\alpha^2\rho_0\rho}{1+\rho_0}\left(2 - 2\rho - \frac{\rho_0}{1+\rho_0}\right)\right]. \tag{3.27}$$

С учетом коэффициента готовности формулы (3.19), (3.21), (3.22), (3.23) и (3.27) принимают соответственно следующий вид:

$$M = \rho\frac{1+\alpha k_\Gamma(1-k_\Gamma)}{k_\Gamma - \rho}, \tag{3.28}$$

$$L = \rho \frac{\rho + (\alpha k_r + 1)(1 - k_r)}{k_r - \rho}, \tag{3.29}$$

$$W = \frac{1 + \alpha k_r (1 - k_r)}{\mu (k_r - \rho)}, \tag{3.30}$$

$$W' = \frac{\rho + (\alpha k_r + 1)(1 - k_r)}{\mu (k_r - \rho)}, \tag{3.31}$$

$$D = \frac{\rho \cdot k_r}{(k_r - \rho)^2} \cdot [1 + \alpha (1 - k_r)(k_r + \rho + \alpha \rho k_r (1 - 2\rho + k_r))]. \tag{3.32}$$

Вероятность того, что в системе находится ровно k заявок при исправном приборе:

$$p_k = \rho \left[p_{k-1} + \lambda_0 \sum_{j=0}^{k-1} \frac{\lambda^{k-j-1}}{(\lambda + \mu_0)^{k-j}} p_j \right] \tag{3.33}$$

и во время восстановления прибора:

$$h_k = \lambda_0 \sum_{j=0}^{k} \frac{\lambda^{k-j}}{(\lambda + \mu_0)^{k-j+1}} p_j. \tag{3.34}$$

Например, вероятность, что в системе находится одна заявка при исправном приборе:

$$p_1 = \rho \left(1 + \frac{\lambda_0}{\lambda + \mu_0}\right) p_0, \tag{3.35}$$

две заявки:

$$p_2 = \rho^2 \left[\left(1 + \frac{\lambda_0}{\lambda + \mu_0}\right)^2 + \frac{\lambda_0 \mu}{(\lambda + \mu_0)^2} \right] p_0, \tag{3.36}$$

три заявки:

$$p_3 = \rho^3 \left[\left(1 + \frac{\lambda_0}{\lambda + \mu_0}\right)^3 + 2\left(1 + \frac{\lambda_0}{\lambda + \mu_0}\right) \frac{\lambda_0 \mu}{(\lambda + \mu_0)^2} + \frac{\lambda_0 \mu^2}{(\lambda + \mu_0)^3} \right] \cdot p_0 \tag{3.37}$$

и т. д.

Вероятность, что в системе находится k заявок при исправном приборе, можно определить также по формуле

$$p_k = \frac{1}{k!} \frac{d^k F_1(x)}{dx^k}\bigg|_{x=0}. \tag{3.38}$$

Аналогично вероятность того, что в системе находится ровно k заявок во время восстановления прибора, определяется по формуле

$$h_k = \frac{1}{k!} \frac{d^k F_2(x)}{dx^k}\bigg|_{x=0}. \tag{3.39}$$

Вероятность того, что в системе находится ровно k заявок независимо от того, исправлен прибор или восстанавливается, можно определить по формуле

$$\bar{p}_k = p_k + h_k = \frac{1}{k!} \frac{d^k F(x)}{dx^k}\bigg|_{x=0}. \tag{3.40}$$

Таким образом, на основе метода производящих функций получена математическая модель функционирования ВС в однопрограммном режиме как одноканальной системы массового обслуживания с ненадежно работающим прибором и неограниченным временем ожидания заявок в очереди на обслуживание.

Полученная математическая модель может быть использована для моделирования обработки данных в однопрограммном режиме с целью получения количественных характеристик прохождения задач через вычислительную систему. Необходимость моделирования может быть иллюстрирована общей постановкой задачи на обработку данных статистической отчетности по промышленности с использованием ЭВМ (табл. 3.1).

Если учесть, что группа задач статистики промышленности всего лишь одна из 17 различных групп, то планирование обработки применительно к конкретному вычислительному центру с целью выбора оптимальной последовательности обработки информации неизбежно требует предварительного моделирования.

Таблица 3.1

Наименование групп	Количество задач	Объем работ за один отчетный период (тыс.)		Время на выполнение обработки (дней)
		суммировок	действий	
Статистика промышленности — всего	133	21076.7	3546.1	1208
в том числе по периодичности:				
декадная	1	3.2	2.3	1
месячная	15	1411.6	733.9	80
квартальная	60	1477.0	763.3	488
полугодовая	4	449.6	37.2	35
годовая	53	17765.3	2009.4	604

При пакетной обработке задач в однопрограммном режиме функционирования ВС возникает необходимость выбора оптимальной последовательности выполнения задач с заданными сроками реализации на одном обслуживающем устройстве. В случае, если возможности ВС являются постоянными, а заявки (задачи пакета), которые должны быть обслужены, уже имеются в наличии, необходимо определить оптимальную в некотором смысле очередность обслуживания заявок.

Проблемы, приводящие к последовательности подобного типа, могут встретиться при планировании выполнения работ на различных вычислительных установках (ЭВМ, машиносчетных станциях, машиносчетных бюро и т. д.). Как и в других проблемах, приводящих к моделям последовательностей, отыскание оптимальной после-

довательности методом перебора достаточно трудоемко при больших n.

Постановка задачи состоит в следующем. Дано n заявок на выполнение работ на одном обслуживающем устройстве к определенному сроку. Для каждой заявки известно:

S_j — длительность выполнения работы j;

C_j — время, к которому должна быть выполнена работа j, или срок выполнения этой работы.

При рассмотрении этой задачи предполагаем, что:

1) время выполнения каждой заявки не зависит от порядка выполнения заявок и известно до начала процесса выполнения;

2) система приоритетов отдельных заявок отсутствует;

3) обслуживающее устройство не может выполнять одновременно более одной заявки. Для данной задачи может быть сформулировано несколько критериев оптимальности:

1) минимальное суммарное отклонение фактического времени выполнения каждой заявки от заданного срока (минимальное отклонение). Этот критерий целесообразно применять в случае невыполнения в срок одной или нескольких заявок;

2) минимальное суммарное просроченное время для заявок, не выполненных в срок (минимальная просрочка);

3) максимальное количество заявок, выполненных к заданному сроку;

4) максимальное суммарное отклонение фактического времени выполнения каждой заявки от заданного срока (максимальный резерв). Этот критерий целесообразно применять в случае, если заявки выполняются к заданным срокам.

Ограничимся рассмотрением решения поставленной задачи по критерию минимального отклонения.

Пусть имеется последовательность заявок:

$$y = 1, 2, \ldots, j-1, j, j+1, \ldots, n.$$

Рассмотрим разность между фактическим временем выполнения заявки и заданным сроком ее реализации. Для заявок, выполняемых в первую очередь:

$$x_1 = S_1 - C_1 = \sum_{j=1}^{1} S_j - C_1,$$

во вторую:

$$x_2 = S_1 + S_2 - C_2 = \sum_{j=1}^{2} S_j - C_2,$$

в третью:

$$x_3 = S_1 + S_2 + S_3 - C_3 = \sum_{j=1}^{3} S_j - C_3.$$

Аналогично

$$x_r = \sum_{j=1}^{r} S_j - C_r.$$

При $x_j \leqslant 0$ заявка выполняется в срок, при $x_j > 0$ — с опозданием. В этом случае выражение для критерия минимального отклонения можно записать так:

$$T_n(y) = \min \sum_{j=1}^{n} x_j = \min_{1 \leqslant r \leqslant n} \sum_{r=1}^{n} \left| \sum_{j=1}^{r} S_j - C_r \right|. \qquad (3.41)$$

Следовательно, задача состоит в определении такой последовательности y, которая будет минимизировать T. Очевидно, что всех возможных последовательностей будет $n!$.

При решении задачи возможны следующие варианты:

1) все заявки не выполняются в срок при любой последовательности выполнения, т. е. $S_j - C_j > 0$ для всех j. Тогда выражение (3.41) можно записать в виде:

$$T_n(y) = \min \left(\sum_{j=1}^{n} (S_j - C_j) + \sum_{j=1}^{n-1} (n-j) S_j \right) \qquad (3.42)$$

и, очевидно, оптимальную последовательность получим в этом случае, выполняя заявки в порядке возрастания времени их реализации;

2) все заявки выполняются в срок при любой последовательности их реализации. В этом случае последовательность реализации заявок должна быть построена так, чтобы максимизировать:

$$T_n(y) = \max \left(\sum_{j=1}^{n} (S_j - C_j) + \sum_{j=1}^{n-1} (n-j) S_j \right). \qquad (3.43)$$

Такую последовательность получим, выполняя заявки в порядке уменьшения времени их реализации;

3) общий случай. Часть заявок не выполняется в срок при любой последовательности реализации. Остальные заявки могут выполняться в срок на первом, втором, ..., n-м месте при любой последовательности.

Для выбора оптимальной последовательности в общем случае используем следующий алгоритм. После определения величин x_j по всем заявкам и установления наличия заявок, не выполняемых в срок, вычислим величину $(|S_j - C_j| + S_j)$ по всем j, кроме тех, для которых выполнено условие

$$\sum_{j=1}^{n} S_j \leqslant C_j, \qquad (3.44)$$

т. е. кроме заявок, которые выполняются в срок при любой последовательности реализации.

Заявкам, для которых это условие не выполнено, необходимо присвоить номера в оптимальной последовательности в порядке увеличения:

$$(|S_j - C_j| + S_j).$$

Оставшимся заявкам присваиваются очередные номера оптимальной последовательности в порядке уменьшения времени реализации этих заявок.

Выполнение указанных действий приводит к определению оптимальной последовательности или последовательности, близкой к оптимальной, что вполне достаточно для практических потребностей.

Рассмотрим следующий пример. Пусть сформирован пакет для обработки данных десяти задач, сведения о которых представлены в табл. 3.2. В этой же таблице приведены расчетные показатели времени завершения обработки и отклонения от времени получения результатов.

Таблица 3.2

Номер задачи j	Время решения S_j	Время получения результатов C_j	Время завершения	Отклонение
1	5	33	5	—28
2	7	35	12	—23
3	9	60	21	—39
4	2	13	23	10
5	3	2	26	24
6	6	26	32	6
7	8	19	40	21
8	4	24	44	20
9	10	59	54	— 5
10	4	3	58	55

Выполним расчеты в рабочей табл. 3.3.

Для задач 3 и 9 условие (3.44) выполняется, и они могут быть выполнены в срок при любой последовательности.

Таблица 3.3

j	S_j	C_j	$S_j - C_j$	$\lvert S_j - C_j \rvert + S_j$	Место в оптимальной последовательности
1	5	33	—28	33	7
2	7	35	—28	35	8
3	9	60	—51		9
4	2	13	—11	13	3
5	3	2	1	4	1
6	6	26	—20	26	6
7	8	19	—11	19	4
8	4	24	—20	24	5
9	10	59	—49		10
10	4	3	1	5	2

Для задач 1, 2, 4 — 8 и 10 составляем оптимальную последовательность в порядке возрастания величины $(|S_j - C_j| + S_j)$.

Для задач 3 и 9 очередность решения устанавливаем в порядке уменьшения времени их реализации.

В полученной последовательности определим суммарное отклонение фактического времени завершения задач от заданного срока.

В заданной последовательности реализации заявок (табл. 3.2) шесть из них были бы выполнены с общим опозданием на 136 единиц времени. В упорядоченной последовательности реализации заявок пять из них (табл. 3.4) будут выполнены с общим опозданием на 11 единиц времени.

Таблица 3.4

Номер задания j	Время решения S_j	Время получения результатов C_j	Время завершения	Отклонение
1	3	2	3	1
2	4	3	7	4
3	2	13	9	— 4
4	8	19	17	— 2
5	4	24	21	— 3
6	6	26	27	1
7	5	33	32	— 1
8	7	35	39	4
9	9	60	48	—12
10	10	59	58	1

3.2. Многопрограммный режим работы

Система работает в многопрограммном режиме, если несколько программ пользователей одновременно находятся в системе и выполнение одной из них может быть прервано для перехода к выполнению другой программы пользователя с последующим возвратом к прерванной программе.

Основной проблемой многопрограммной работы является организация защиты программ от взаимного влияния как на уровне оперативной памяти, так и на разных уровнях внешней памяти. Необходимость организации защиты возникает вследствие того, что различные программы пишутся независимо друг от друга в расчете на единоличное использование ресурсов системы. Это создает опасность взаимного влияния программ друг на друга и должно быть учтено при реализации системы.

Разделение аппаратных и программных ресурсов системы ставит сложные задачи управления перед СУПЕРВИЗОРом, которые он решает благодаря наличию специальных алгоритмов распределения ресурсов системы.

Принято различать три категории правил перехода от программы к программе и соответственно три типа многопрограммной работы: классическое мультипрограммирование, параллельная обработка, разделение времени.

Режим мультипрограммирования характеризуется тем, что правила перехода от программы к программе устанавливаются из соображений достижения максимальной производительности машины путем более широкого использования совмещений. Термин «совмещение» используется как для характеристики действия систем, включающих несколько устройств обработки информации, работающих в одно и то же время, так и для описания режима, когда попеременное выполнение нескольких программ короткими отрезками времени создает у пользователя впечатление одновременности.

Первая форма совмещения реализуется определенной множественностью элементов на уровне оборудования, например обеспечение ввода-вывода по нескольким параллельным каналам с одновременным выполнением вычислительного процесса в центральном устройстве.

Вторая форма совмещения реализуется, как правило, программно. Исходя из того, что основной целью мультипрограммирования является достижение полной загрузки всех устройств машины, не следует на данном этапе фиксировать внимание на той или иной форме совмещения: главное — использовать полностью обе формы.

При мультипрограммировании улучшение качества обслуживания пользователя непосредственно не предусматривается. Может случиться так, что одна из программ будет редко оказываться в состоянии ожидания и блокирует выполнение других программ. Кроме того, мультипрограммирование не совместимо с непосредственным доступом, так как это один из видов пакетной обработки задач, т. е. все пользователи получают результаты одновременно, как и в режиме косвенного доступа. Тем не менее общее время обработки пакета задач при мультипрограммировании оказывается меньше, чем при последовательной пакетной обработке, за счет лучшего использования оборудования.

Производительность системы, работающей в мультипрограммном режиме, во многом зависит от состава группы программ, исполняемых совместно. Действительно, почти неизбежно возникновение отдельных моментов времени, когда все программы ожидают ввод и вывод данных, а центральное устройство простаивает. Отсюда следует задача целенаправленного формирования пакета программ с целью сокращения непроизводительных простоев оборудования.

Под *режимом параллельной обработки данных* нескольких программ понимается такой многопрограммный режим, в котором переход от одной программы к другой происходит через достаточно короткие промежутки времени, сравнимые со скоростью машины, чтобы создать у пользователя впечатление одновременности исполнения нескольких программ (режим кажущегося совмещения).

Основной целью данного режима является улучшение обслужи-

вания пользователей, выражающееся в том, что у последних создается впечатление отсутствия очереди, так как решение их задач не прерывается на длительные отрезки времени. Кроме того, если пользователю предоставляются некоторые средства прямого доступа (хотя бы для вывода информации), то это впечатление еще более усиливается выдачей результатов по мере их получения.

Однако в общем случае время ответа при этом режиме не оптимально, так как из-за отсутствия соответствующих правил приоритета обслуживания задач выбор последних производится сканированием, а время выполнения пакета задач при параллельной обработке значительно превышает время их выполнения при единоличном использовании машины. Тем не менее параллельная обработка данных ряда задач, при которой пользователь может получать результаты, не ожидая конца обслуживания всего пакета, уменьшает время ожидания ответа по сравнению с режимом последовательной обработки.

В большинстве случаев параллельная обработка сочетается с мультипрограммированием, что обеспечивается наличием двух типов прерываний, вызывающих переход к другой программе: естественное прерывание во время ожидания ввода-вывода (мультипрограммирование); вынужденное прерывание в конце отрезка времени, отводимого каждой программе (параллельная обработка).

В настоящее время все большее внимание уделяется восстановлению возможности диалога человек — машина в варианте непосредственного доступа, лишенном основных недостатков взаимодействия, которые были отмечены выше. Речь идет о форме эксплуатации с разделением времени, когда пользователи включаются на линию с ЭВМ через терминальные устройства, представляющие собой, например, обычную пишущую машинку, и работают с ними как с настоящим пультом управления. Так как имеет место коллективный доступ к ЭВМ, то время реакции пользователя не является потерей для системы, которая может быть загружена обработкой запросов от других пользователей.

Разделение времени — наиболее развитая форма многопрограммной работы, совмещающая мультипрограммирование с непосредственным доступом и работой в реальном времени для всех пользователей. Время ответа близко к тому, которое было при единоличном использовании машины.

Как и при параллельной работе, программы выполняются поочередно в течение некоторого отрезка времени, по завершении которого происходит вынужденное прерывание. Длительность этих отрезков времени (квантов) не является, как правило, фиксированной, а изменяется в зависимости от рассматриваемой программы, а также от конкретных условий эксплуатации в момент передачи управления от одного пользователя другому. Выбор пользователя, программе которого будет передано управление, и выделение этой программе кванта времени осуществляет планирующая программа, являющаяся составной частью программ-диспетчеров, предназначенных для работы с разделением времени.

Если задача не решилась полностью в отведенный ей квант, то она прерывается и перемещается в конец очереди заявок, ожидающих дальнейшей обработки, причем в процессе смены задач некоторое время работы центрального процессора (время переключения — r) непроизводительно расходуется на обмен между ступенями памяти и работу служебных программ. При одной общей очереди заявок с одинаковой приоритетностью подобный алгоритм осуществляет стратегию циклического планирования.

Дальнейшими модификациями этого метода планирования являются многоприоритетные алгоритмы планирования, позволяющие сократить потери времени на обмен. При такой стратегии планирования новые заявки поступают в очередь высшего приоритета. Если в представленный квант времени задача не закончена, то она перемещается в конец очереди более низкого приоритета и вызывается для исполнения только при отсутствии заявок в очередях с большими приоритетами. Многоприоритетные алгоритмы планирования обеспечивают быструю реакцию на запросы, связанные с коротким временем обслуживания, не слишком задерживая при этом выполнение длинных программ и обеспечивая тем самым оптимизированное время ответа.

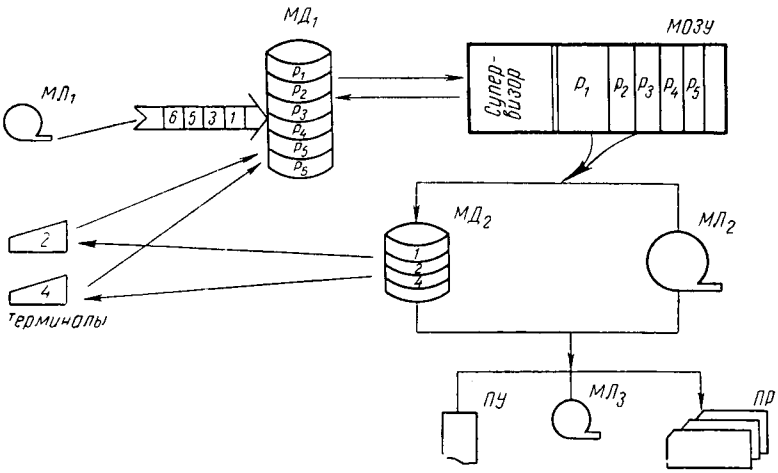


Рис. 3.4. Схема режима пакетной обработки

Рассмотрим ряд форм эксплуатации и типичные структуры систем с разделением времени, которые наиболее часто встречаются на практике.

Системы, работающие в режиме пакетной обработки (рис. 3.4), сочетают две различные формы эксплуатации. Первая, преобладающая, полностью исключает непосредственную связь пользователя с машиной. Как и в режиме последовательной обработки, осуществляется формирование и ввод пакетов программ на магнитную ленту

(МЛ), которая далее служит входной лентой для системы. С этой ленты программы переписываются в промежуточную память (МД и МБ), а затем исполняются в многопрограммном режиме. Результаты решения задач выводятся по мере получения в промежуточную память типа магнитных лент ($МЛ_2$) или магнитных дисков ($МД_2$).

По мере освобождения внешних устройств (печатающее устройство, перфораторы, магнитная лента и т. д.) результаты решения задач выводятся на эти устройства и в таком виде выдаются пользователю по окончании обработки всего пакета задач.

Вторая форма эксплуатации сочетает *режим пакетной обработки ряда пользователей с привилегированным режимом* включения в обработку некоторых удаленных пользователей. Программы привилегированных пользователей помещаются, как и в предыдущем режиме, до начала исполнения в память на дисках ($МД_1$) и обладают более высоким приоритетом по сравнению с программами, обрабатываемыми пакетно. Результаты решения задач также переписываются по мере их получения в промежуточную память, но привилегированный пользователь получает их сразу после завершения его программы. Таким образом, время ответа для привилегированных пользователей может быть достаточно удовлетворительным при исключении их непосредственного диалога с машиной.

Программы, размещенные в промежуточной памяти на дисках, исполняются в соответствии с правилами классического мультипрограммирования. В связи с тем, что объем оперативной памяти недостаточен для хранения всех ждущих программ, необходимо предусмотреть обмен информацией между оперативной и внешней памятью. Однако этот обмен стремятся по возможности сократить до минимума и допускать присутствие в оперативной памяти нескольких программ или частей нескольких программ, отличных от той, которая исполняется в данный момент.

Один из путей сокращения времени обмена информацией между оперативной и внешней памятью — уменьшение частоты обменов.

Первая возможность уменьшения частоты обменов состоит в уменьшении частоты передач управления от одной программы к другим, т. е. в увеличении кванта времени, отводимого задаче. При этом возникает опасность блокировки программ до истечения выделенного отрезка времени, например при ожидании ввода-вывода. Кроме того, недостаточно частое прерывание программ увеличивает среднее время реакции системы на запрос пользователя.

Вторая возможность уменьшения частоты обменов заключается в том, чтобы передача управления от программы к программе не приводила бы к обязательному обмену с внешней памятью. С этой целью стремятся иметь в оперативной памяти некоторую часть программ в состоянии ожидания. Недостатком этого решения является то, что при ограниченном объеме оперативной памяти предусматривается его нерациональное расходование под ждущие программы.

Среди других возможностей уменьшения частоты обменов можно отметить выполнение программ по блокам. Переключение с блока на блок в этом случае происходит с помощью аппаратных средств системы либо СУПЕРВИЗОРа, реагирующего на специально расставленные при трансляции инструкции по точкам деления программы на блоки. Для того чтобы метод разбиения программ был достаточно эффективным, необходимо обеспечить выполнение отдельного блока в течение определенного времени без обращения к другим блокам. Это влечет за собой необходимость размещения констант и рабочих ячеек в самом блоке.

Однако несмотря на все приемы сокращения количества обменов информацией между оперативной и внешней памятью, обмены при разделении времени занимают существенное место. Отсюда возникает необходимость уменьшения потери времени обменов для системы путем перераспределения его другим пользователям при условии наличия в оперативной памяти места для размещения их программ.

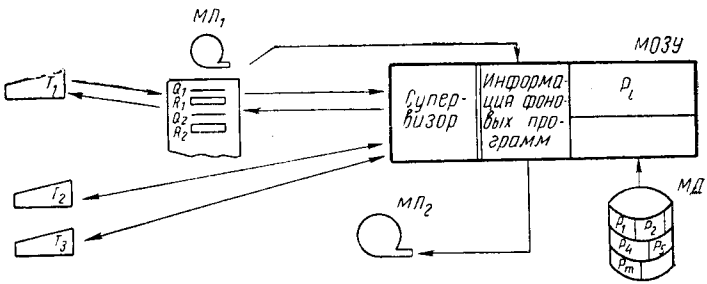


Рис. 3.5. Схема режима «запрос-ответ»

При работе в *режиме «запрос — ответ»* предусматривается наличие программ информационно-поисковых систем (ИПС), призванных обеспечить оперативной информацией по запросам. В свободное от запросов время система выполняет некоторую фоновую работу, например пакетную обработку задач (рис. 3.5). Фоновую работу составляют задания, время выполнения которых велико по сравнению с временем решения задач с разделением времени. Эти задания наделяются низшим приоритетом в системе с разделением времени и запускаются с пульта ЭВМ, рассматриваемого как удаленный пульт пользователя.

Пользователи ИПС со своих пультов-терминалов (Т) имеют прямой доступ к системе и могут послать в кодированной форме конечную серию вопросов Q_i $(i=1, 2, \ldots, m)$. Каждому из вопросов Q_i соответствует своя программа системы P_i, вырабатывающая соответствующий ответ R_i. Группа отвечающих программ P_i постоянно хранится в промежуточной памяти типа дисков в виде некоторой библиотеки модулей.

Когда на одном из пультов задается вопрос Q_i, СУПЕРВИЗОР прерывает фоновую задачу, осуществляет поиск и вызов программы P_i в оперативную память и отдает ей управление. Программа P_i вырабатывает ответ R_i и выдает его на пульт пользователя. Если нет других запросов, возобновляется решение фоновой задачи. Гибкость системы ограничивается конечностью допустимых вопросов. Такая система обычно располагает возможностью дальнейшего развития.

При работе *в разговорном режиме* предусматривается разделение ресурсов системы между ограниченным количеством удаленных пользователей, имеющих прямой доступ к машине через терминалы (рис. 3.6).

В распоряжение пользователей предоставляется только один язык программирования, чаще всего это разговорный язык, позволяющий пошаговую трансляцию, выдачу содержимого ячеек памяти в любой момент и т. п. Для получения небольшого времени ответа, необходимого для ведения диалога, как транслятор, так и программы пользователей должны постоянно находиться в фиксированных полях оперативной памяти.

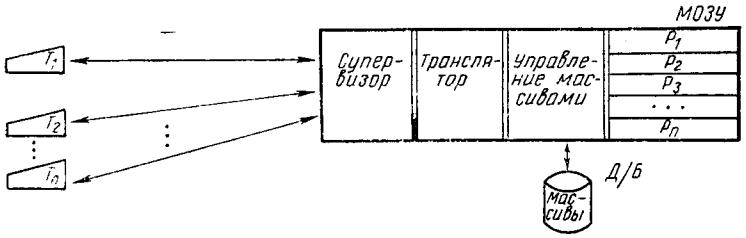


Рис. 3.6. Схема работы в разговорном режиме

Для хранения массивов пользователей выделяется промежуточная память на дисках или барабанах. Хранение информации пользователей в архиве ограничивается конечным отрезком времени. Исходя из принципа минимизации времени реакции системы на запросы пользователей не рекомендуется частое обращение к архивам, что значительно ограничивает возможности и пользователей, и системы.

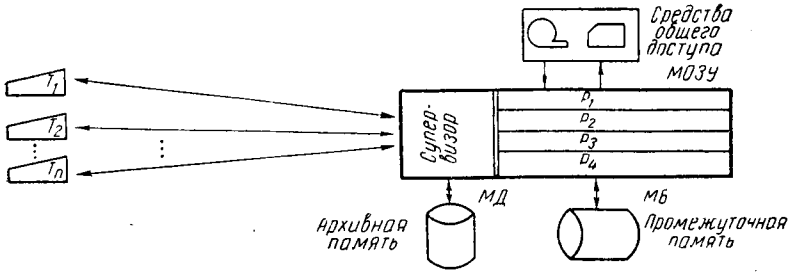


Рис. 3.7. Схема работы в универсальном режиме

Системы, работающие в универсальном режиме, обладают всеми возможностями рассмотренных выше режимов (рис. 3.7). Пользователи имеют прямой доступ к машине через удаленные пульты и, кроме того, обладают набором средств общего доступа, расположенных в непосредственной близости от машины.

Программы и данные пользователя могут поступать в системы извне через пульты или средства общего доступа или получаться в результате работы какой-либо программы. Пользователь осуществляет вход в систему, ввод информации и запуск своих программ с помощью специального языка управления.

Команды, поступающие от различных пультов, выполняются совместно в соответствии с правилами разделения времени. Программы, обслуживающие различных пользователей, могут одновременно находиться в оперативной памяти. Естественно, что они присутствуют в этой памяти только отдельными сегментами. Вспомогательная, или промежуточная память, выполненная в виде быстрого барабана, хранит в состоянии ожидания не поместившиеся в оперативной памяти активные программы, т. е. программы, запрошенные для исполнения с одного из пультов.

Для достижения большей производительности при работе в универсальном режиме в систему включают несколько независимых процессов, взаимодействующих через оперативную память или специальный канал связи. Возможный вариант такой системы с двумя процессорами приведен на рис. 3.8.

Подобная организация привела к *мультипроцессорным вычислительным системам,* способным параллельно выполнять несколько независимых программ или несколько независимых ветвей одной программы. Высокие требования к надежности и живучести ВС предопределили наличие в ней как минимум двух процессоров и возможность двухмаршрутной связи между модулями системы.

Свойства мультипроцессорных ВС отражают две основные тенденции современного развития вычислительной техники:

1) модульный принцип построения, при котором каждое устройство выполняется в виде нескольких независимых модулей, что позволяет наращивать структуру и делает систему менее уязвимой к отказам в силу взаимозаменяемости однотипных модулей;

2) параллельное выполнение независимых ветвей одной программы (параллельное выполнение полностью независимых программ), что позволяет уменьшить общее время реализации задач.

Второе свойство отличает мультипроцессорные системы от мультипрограммных, в памяти которых хранится несколько программ, совмещаются работы внешних и центральных устройств и выполнение отдельных операций в центральном устройстве. Основная цель мультипрограммирования — наиболее полное использование оборудования — лучше всего достигается при непрерывном обращении в системе большого количества программ, когда, несмотря на прерывания, и, следовательно, некоторое увеличение времени выполне-

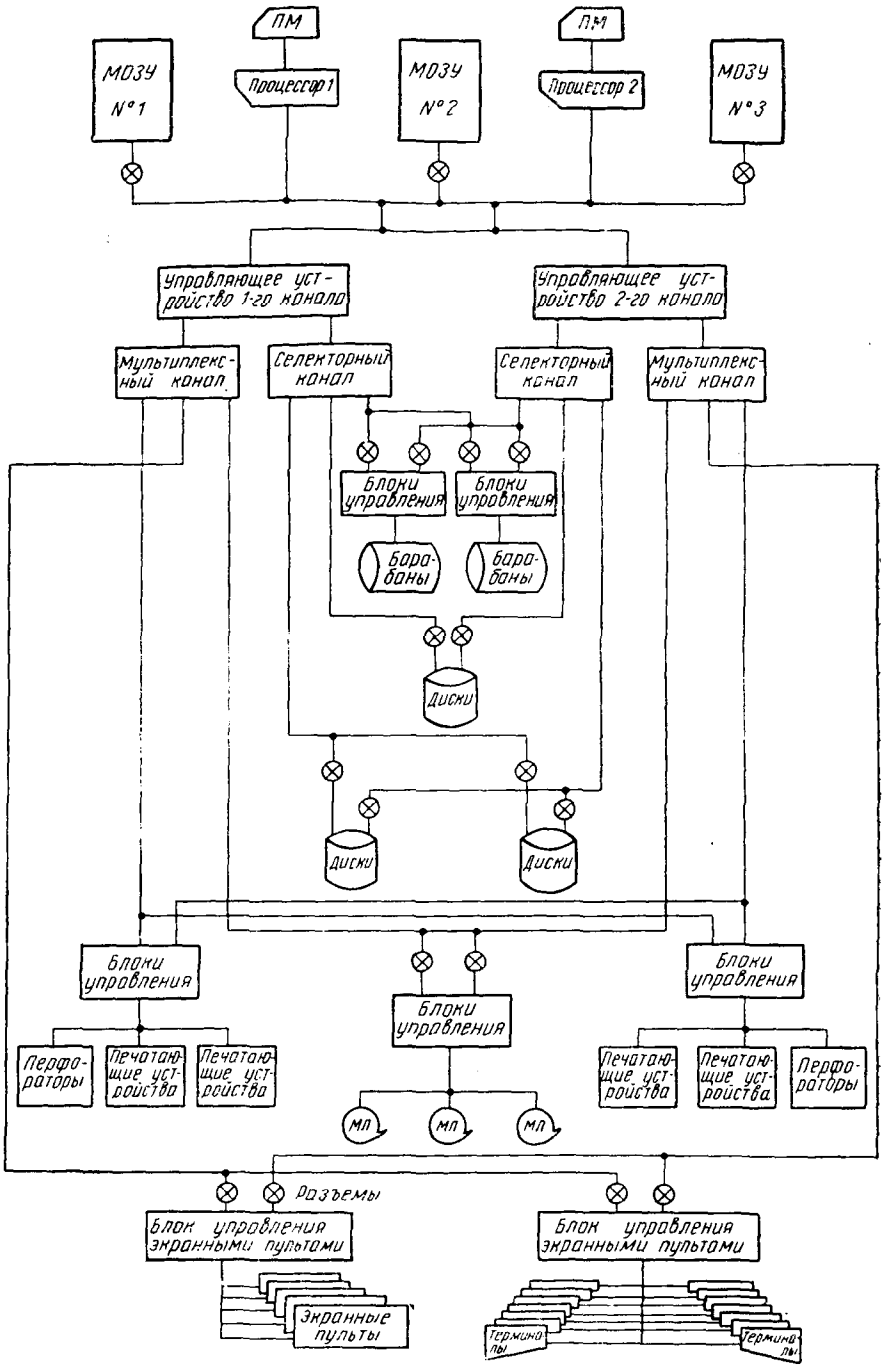


Рис. 3.8. Вариант структуры двухпроцессорной ВС

ния одной программы, время решения многих задач уменьшается по сравнению с временем их решения на машине без совмещения операций.

Мультипроцессорная система в отличии от мультипрограммной позволяет уменьшить время решения каждой отдельной задачи за счет одновременного выполнения независимых участков ее программы и достигнуть высокого коэффициента использования оборудования при непрерывной обработке большого количества программ в универсальном режиме.

Дисциплины обслуживания заявок

При определении эффективности многопрограммного режима работы ВС, кроме собственной эффективности алгоритмов переработки информации, существенное значение имеют эффективность диспетчеризации решения основных функциональных задач, приема и обработки сообщений пользователей, а также подготовка и выдача команд потребителям информации.

Для изучения различных алгоритмов диспетчеризации вычислений в многопрограммном режиме работы целесообразно рассматривать работу ВС как функционирование одноканальной системы массового обслуживания с неограниченным временем ожидания заявок в очереди. Под заявками понимаются сообщения или любые другие требования на включение в работу задач (программ) пользователей или системных программ, а под процессом обслуживания заявок — выполнение этих программ машиной.

Эффективность методов диспетчеризации определяется возможностью задержки или потери заявки перед обработкой. В зависимости от типа управляющей системы задержка заявок может учитываться либо по средней величине времени задержки, либо по величине допустимого времени ожидания. Соответственно предполагается, что сообщение или заявка обесцениваются либо пропорционально времени задержки, либо скачком при превышении некоторого допустимого времени ожидания. В первом случае систему диспетчеризации можно оценить с помощью функционала:

$$C = \sum_{i=1}^{m} \alpha_i \lambda_i W_i,$$

где α_i — штраф за единицу времени ожидания заявки i-го типа;
λ_i — интенсивность i-го потока заявок;
W_i — средняя длительность ожидания в очереди i-х заявок;
m — количество типов заявок.

Значение C представляет собой величину среднего суммарного штрафа за единицу времени функционирования ВС вследствие ожидания заявок.

В тех случаях, когда установлено время ожидания в пределах

τ_{0i}, критерий эффективности диспетчеризации вычислений может быть записан в виде:

$$C' = \sum_{i=1}^{m} \alpha_i' \lambda_i p\,(>\tau_{0i}),$$

где α_i' — штраф за потерю одной заявки i-го типа;
$p(>\tau_{0i})$ — вероятность ожидания заявки i-го типа свыше допустимого времени τ_{0i}.

Потеря заявок до обработки, кроме допустимого времени ожидания, определяется допустимым объемом буферной памяти. При переполнении памяти заявки i-го типа теряются с вероятностью p_i и потери при данной системе диспетчеризации определяются функционалом:

$$C'' = \sum_{i=1}^{m} \alpha_i'' \lambda_i p_i,$$

где α_i'' — штраф за потерю одной заявки i-го типа из-за переполнения буферной памяти.

В реальных системах эффективность диспетчеризации редко определяется одним из приведенных функционалов, как правило, требуется совместная оценка ожидания и потери заявок, т. е. потери оцениваются суммой:

$$C_\Sigma = C + C' + C''.$$

В основе приведенных оценок лежат предположения об аддитивности потерь для различных типов заявок и о линейной зависимости потерь от времени задержки. В некоторых случаях определение эффективности диспетчеризации резко усложняется в силу сложной зависимости штрафа от времени ожидания и потери заявки.

Таким образом, при выборе метода диспетчеризации вычислений возникает задача минимизации возможных потерь информации путем построения рациональных дисциплин заполнения буферной памяти, выбора данных для обработки и выдачи информации внешним абонентам. При анализе систем по вероятности потери сообщений необходима оценка минимально допустимого объема буферной памяти. Естественно ожидать, что более эффективные системы диспетчеризации являются и более сложными в реализации, т. е. требуют большого объема команд и времени счета. Все это может отрицательно сказаться на эффективности многопрограммного режима функционирования ВС.

При изучении дисциплины обслуживания заявок предполагается, что процессы поступления и обслуживания являются независимыми. Поступающая заявка начинает немедленно обслуживаться, если в этот момент свободна машина. Если же машина занята обслуживанием ранее поступившей заявки, то в зависимости от типа поступившей и обслуживаемой заявки, а также от вида используемого алгоритма диспетчеризации поступившая заявка может либо ожидать своей очереди на обслуживание, либо прервать выполняе-

мую программу. В том случае, когда происходит прерывание обслуживания заявки, предполагается, что эта заявка возвращается обратно в очередь, где и ожидает возобновления или продолжения прерывного обслуживания. Таким образом, длительность пребывания заявки в памяти машины складывается из времени ожидания заявки в очереди и времени ее обслуживания вычислительной машиной.

Наиболее характерные дисциплины обслуживания заявок представлены на рис. 3.9:

бесприоритетные дисциплины обслуживания, при которых выбор заявок из числа ожидающих в очереди на обслуживание производится в некотором заранее установленном порядке без учета их относительной важности и времени обслуживания;

приоритетные дисциплины обслуживания, при реализации которых отдельным заявкам предоставляется преимущественное право на обслуживание перед другими заявками.

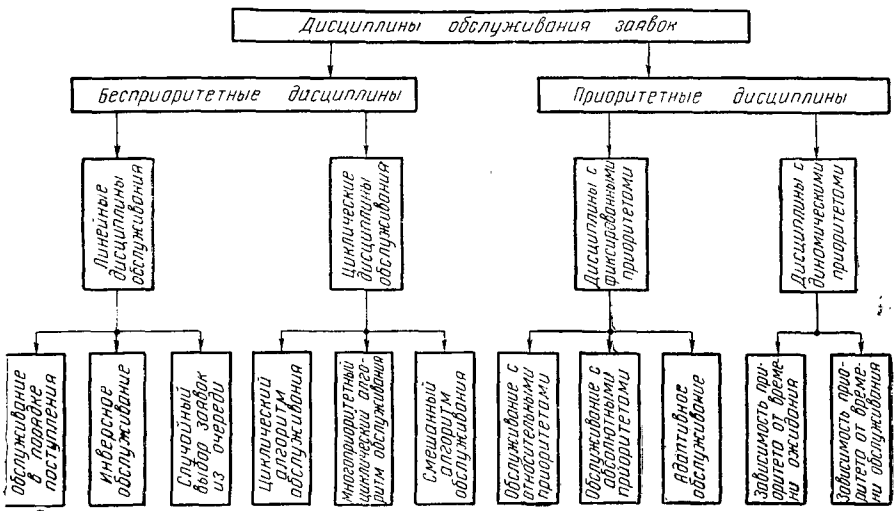


Рис. 3.9. Схема классификации дисциплин обслуживания заявок

Бесприоритетные дисциплины обслуживания

Для наиболее простого случая, когда в ЭВМ поступает пуассоновский поток однотипных заявок с интенсивностью λ, а длительность обслуживания заявок характеризуется произвольной функцией распределения $B(t)$, единственным признаком для выбора дисциплины обслуживания является последовательность поступления заявок в машину. Эта последовательность может однозначно определять очередность обслуживания заявок, учитываться частично или полностью игнорироваться. Среди бесприоритетных дисциплин обслуживания можно выделить линейные и циклические дис-

циплины. В классификационной схеме (рис. 3.9) представлены возможные варианты бесприоритетных линейных дисциплин обслуживания: обслуживание в порядке поступления; инверсное обслуживание (первой обслуживается заявка, поступившая последней); случайный выбор заявок из очереди. Ограничимся рассмотрением показателей только последовательной дисциплины обслуживания.

При экспоненциальном распределении времени обслуживания функция распределения длительности ожидания заявок в очереди для рассматриваемой дисциплины имеет вид:

$$H(t) = 1 - \rho \cdot e^{-(\mu-\lambda)\cdot t}. \qquad (3.45)$$

Среднюю длительность ожидания заявок в очереди можно определить:

$$W = \int_0^\infty t\,dH(t) = \frac{\rho}{\mu(1-\rho)}.$$

Необходимо располагать и такими характеристиками, как возможный разброс фактических значений длительности ожидания относительно среднего значения. Для этого используют дисперсию, среднеквадратическое отклонение $\sqrt{D_W}$ и коэффициент вариации V_W длительности ожидания заявок в очереди.

Интегрируя выражение (3.45), находим второй начальный момент длительности ожидания:

$$W_2 = \int_0^\infty t^2 dH(t) = \frac{2\rho}{\mu^2(1-\rho)^2}.$$

Дисперсия длительности ожидания заявок в очереди определяется по разности между начальными и центральными моментами:

$$D_W = W_2 - W^2 = \frac{\rho(2-\rho)}{\mu^2(1-\rho)^2}.$$

Среднеквадратическое отклонение и коэффициент вариации длительности ожидания можно определить следующими зависимостями:

$$\sqrt{D_W} = \frac{\sqrt{\rho(2-\rho)}}{\mu(1-\rho)},$$

$$v_W = \frac{\sqrt{D_W}}{W} = \sqrt{\frac{2-\rho}{\rho}}.$$

В ряде случаев необходимо оценивать продолжительность пребывания заявок в системе, т. е. длительность интервала времени от момента поступления заявки в очередь до момента завершения ее обслуживания. При экспоненциальном распределении времени обслуживания среднее значение этой величины можно определить из выражения:

$$V = W + \frac{1}{\mu} = \frac{1}{\mu(1-\rho)}.$$

Отметим, что не все поступающие заявки ожидают в очереди начала обслуживания, так как часть из них поступает в такие моменты времени, когда система свободна, и поэтому они немедленно принимаются к обслуживанию. Вероятность того, что в момент поступления заявки система занята обслуживанием:

$$p(>0)=\rho,$$

поэтому средняя длительность ожидания заявок в очереди:

$$W(>0)=\frac{W}{P(>0)}=\frac{1}{\mu(1-\rho)}.$$

На основе этого выражения можно определить среднее количество заявок в системе:

$$L=\lambda\cdot V=\frac{\rho}{1-\rho}.$$

Учитывая, что математическое ожидание числа заявок, находящихся на обслуживании в произвольный момент времени, равно ρ, можно найти среднее число заявок в очереди:

$$L_0=L-\rho=\frac{\rho^2}{1-\rho}.$$

Для оценок характеристик алгоритмов диспетчеризации в ряде случаев оказывается необходимым знать вероятности задержки информации свыше некоторого заданного времени, а также вероятности одновременного пребывания в памяти машины определенного количества заявок. Эти показатели могут быть получены при наличии выражения для функции распределения времени ожидания в очереди $H(t)$ и распределения длины очереди P_m.

Из циклических дисциплин обслуживания, имеющих сравнительно большее практическое распространение в системах разделения времени (СРВ), рассмотрим: циклический алгоритм с одной очередью; циклический алгоритм с N очередями; смешанный алгоритм обслуживания.

Анализ рассматриваемых алгоритмов планирования обслуживания заявок проводится при следующих ограничениях:

1) потоки от нескольких пользователей аппроксимируются одним пуассоновским потоком с интенсивностью $\lambda=\text{const}$;

2) время обслуживания поступающих заявок, определяемое длительностью обработки в центральном устройстве при однопрограммном режиме, распределяется экспоненциально с функцией распределения:

$$B(t)=P(S\leqslant t)=1-e^{-\mu t};$$

3) используется дисциплина квантового обслуживания в порядке поступления заявок с возвратом прерванной заявки в начало очереди;

4) потери времени на переключение обслуживания заявок постоянны ($r=\text{const}$);

5) длина очереди заявок не ограничена;

6) рассматривается стационарный режим при отсутствии перегрузок.

Основной целью анализа является нахождение аналитических выражений, определяющих среднее время ожидания в очереди задач, классифицированных по длительности их реализации, длину очереди и ряд других характеристик. Организация эффективного взаимодействия пользователей с системой осуществляется с помощью алгоритмов планирования, распределяющих время центрального процесса между несколькими задачами.

Циклический алгоритм планирования с одной очередью

При циклическом алгоритме планирования заявки обслуживаются центральным вычислителем (ЦВ) в порядке их поступления, каждая в течение определенного кванта времени. Если программа выполнена в течение этого кванта времени, то результаты выводятся, и на обработку поступает следующая программа, в противном случае незаконченная программа ставится в конец очереди и ожидает следующего кванта обслуживания. Схема функционирования модели представлена на рис. 3.10.

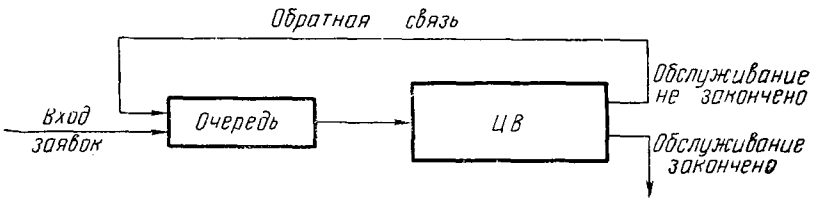


Рис. 3.10. Схема циклического алгоритма с одной очередью

Пусть максимальное время кванта составляет величину Q, а время решения некоторой задачи — S. Тогда величина K, получаемая из неравенства

$$(K-1)\,Q \leqslant S < KQ,\ K = 1, 2, \ldots, \qquad (3.46)$$

будет определять необходимое целое число квантов для обработки информации данной задачи и является признаком классификации заявок входного потока. Обозначим через τ_i время цикла, проходящее между моментами окончания $(i-1)$-го кванта и начала i-го кванта обслуживания данной заявки. Тогда время ожидания заявки в очереди составит величину

$$W_K = \sum_{i=1}^{k} \tau_i. \qquad (3.47)$$

С учетом времени переключения величина расширенного кванта может быть определена как

$$Q_p = Q + r. \qquad (3.48)$$

На основании предположений 2—4 функция распределения длительности K квантов обслуживания имеет вид:

$$F_K(t)=\begin{cases}0 \text{ при } (j-1)\,Q_p \leqslant t < r+(j-1)\,Q_p,\\ 1-e^{-\mu(t-jr)} \quad \text{при } r+(j-1)\,Q_p \leqslant t < jQ_p,\\ 1 \text{ при } t \geqslant KQ_p,\ j=1,\ 2,\ \ldots,\ K.\end{cases} \tag{3.49}$$

В дальнейших рассуждениях будем использовать первый и второй моменты функции распределения $F_K(t)$. Введя обозначение

$$\varepsilon = e^{-\mu Q},$$

и опуская громоздкие промежуточные выкладки, запишем:

$$M_{1K}=(1-\varepsilon^K)\left(\frac{1}{\mu}+\frac{r}{1-\varepsilon}\right), \tag{3.50}$$

$$M_{2K}=\frac{2}{\mu^2}(1-\varepsilon^K-K\mu Q\varepsilon^K+\Theta_K), \tag{3.51}$$

где

$$\Theta_K = r\cdot\frac{Q\mu^2[\varepsilon-\varepsilon^K(\varepsilon+K-K\varepsilon)]+\mu(1-\varepsilon)[1-\varepsilon^K(1+K-K\varepsilon)]+\frac{\mu^2 r}{2}[(1+\varepsilon)(1-\varepsilon^K)-2K\varepsilon^K(1-\varepsilon)]}{1-\varepsilon^2}.$$

Рассмотрим случай поступления в систему в некоторый момент заявки, не получившей ни одного кванта обслуживания. Можно предположить, что до этого в системе находилось среднее количество заявок $\overline{n}_1$, а процессор с вероятностью δ занят обслуживанием первой из этих заявок. Средняя длительность обслуживания (средняя длина расширенного кванта) определяется как первый момент $F_K(t)$ при $K=1$ и находится из формулы

$$\overline{Q}_p=(1-\varepsilon)/\mu+r. \tag{3.52}$$

Первое слагаемое $(1-\varepsilon)/\mu$ представляет собой время обслуживания заявки в течение кванта, т. е. $\overline{Q}=(1-\varepsilon)/\mu$.

Начало обслуживания очередной заявки и случайные моменты времени поступления в систему новых заявок независимы.

В связи с этим функция распределения времени обслуживания заявки, обрабатываемой процессором в момент входа новой заявки в систему, выражает фактически распределение остатка кванта обслуживания. Средняя длительность остатка кванта обслуживания определяется:

$$\overline{Q}_1=\frac{1}{1-\varepsilon}\left[\frac{1}{\mu}(1-\varepsilon)-Q\varepsilon\right].$$

Время первого цикла обслуживания:

$$\tau_1=\overline{n}_1\overline{Q}_p-\delta(\overline{Q}-\overline{Q}_1)=\overline{n}_1\overline{Q}_p-\delta\beta, \tag{3.53}$$

где

$$\beta=\frac{\varepsilon}{1-\varepsilon}\left[Q-\frac{1}{\mu}(1-\varepsilon)\right]. \tag{3.54}$$

Среднее количество заявок в системе с разделением времени получим на основании зависимостей (3.50), (3.51) и формулы Полячека-Хинчина:

$$\bar{n}_1 = \frac{\lambda}{\mu}\left[1 + \frac{\mu \cdot r}{1-\varepsilon} + \frac{\lambda(1+\Theta)}{\mu - \lambda - \dfrac{\mu\lambda r}{1-\varepsilon}}\right], \tag{3.55}$$

где

$$\Theta = r \cdot \frac{\mu^2 Q\varepsilon + \mu(1-\varepsilon) + \dfrac{\mu^2 r}{2}(1+\varepsilon)}{(1-\varepsilon)^2}.$$

Вероятность занятости центрального вычислителя определяется как $\lambda \cdot M_{1K}$ при $K=\infty$. Тогда величина

$$\delta = \frac{\lambda}{\mu} + \frac{\lambda r}{1-\varepsilon} \tag{3.56}$$

представляет собой фактическую загрузку системы с разделением времени, совпадающую с $\rho(\rho = \frac{\lambda}{\mu})$ при $r=0$.

Если время решения задачи превосходит величину отведенного кванта обслуживания, то в момент второго поступления заявки в очередь перед ней будут находиться $\bar{n}_2$ заявок. Так как после срабатывания обратной связи в момент i-го $(i=2,3,\ldots)$ поступления заявки в очередь определяется момент начала обработки очередной заявки, в этом случае $Q_1=Q$ и $\beta=0$. Величину n_2 можно определить как сумму среднего количества новых заявок, поступивших в систему за время (τ_1+Q_p) и среднего количества частично обслуженных заявок, возвращающихся в очередь. Используя теорему Литтла и основное свойство экспоненциального распределения, находим выражение для вычисления τ_2:

$$\tau_2 = \tau_1\alpha + Q_p(\alpha - \varepsilon) + \delta\beta\varepsilon,$$

где

$$\alpha = \rho + \varepsilon(1-\rho) + \lambda r.$$

Рассуждая аналогично и применяя метод математической индукции, можно получить:

$$\tau_i = \tau_{i-1}\alpha + Q_p(\alpha - \varepsilon), \quad i = 3, 4, \ldots$$

Общее выражение для времени цикла получим, вычисляя последовательно τ_i через $\tau_{i-1}, \tau_{i-2}, \ldots, \tau_{i-j}$ $(i-j \geqslant 2)$:

$$\tau_i = \begin{cases} \bar{n}_1\bar{Q}_p - \delta\beta & \text{при } i = 1, \\ \bar{n}_1\bar{Q}_p\alpha^{i-1} - \delta\beta\alpha^{i-1} + Q_p(\alpha-\varepsilon)(1-\alpha^{i-1})/(1-\alpha) + \delta\beta\alpha^{i-2} & \text{при } i = 2, 3, \ldots \end{cases} \tag{3.57}$$

Подставляя (3.57) в (3.47), получим окончательное выражение для математического ожидания времени пребывания в очереди заявок K-го класса:

$$W_K = \frac{1-\alpha^K}{1-\alpha}(\overline{n_1}\overline{Q}_p - \delta\beta) + \frac{KQ_p(\alpha-\varepsilon)}{1-\alpha} - \frac{Q_p(\alpha-\varepsilon)(1-\alpha^K)}{(1-\alpha)^2} + \\ + \delta\beta\varepsilon\frac{1-\alpha^{K-1}}{1-\alpha}. \qquad (3.58)$$

Среднее время, проходящее от момента окончания запроса на решение задачи до начала выдачи ее результатов (время отклика системы с разделением времени), определяется выражением

$$V = W_K + S.$$

Приведенные зависимости позволяют количественно определить характеристики циклического алгоритма с одной очередью.

Многоприоритетный циклический алгоритм обслуживания

В отличие от предыдущего алгоритма здесь организуется N очередей (рис. 3.11).

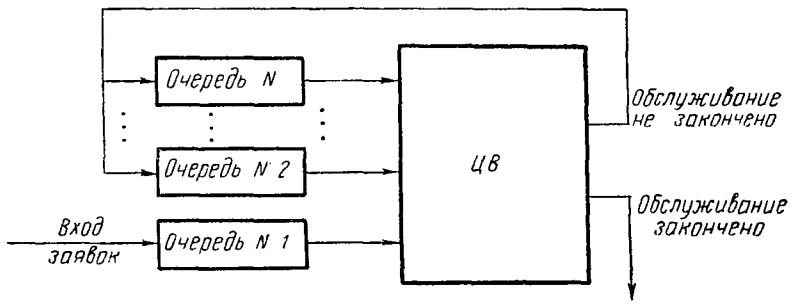


Рис. 3.11. Схема циклического алгоритма с очередями

Для очередей устанавливаются приоритеты, первая очередь обладает наивысшим приоритетом. Внутри очереди приоритетность заявок одинакова.

Новая заявка становится в очередь наивысшего приоритета. Если при предоставлении заявке очередного кванта обслуживание не завершено, то она перемещается в конец очереди более низкого приоритета. Заявки из последней очереди обрабатываются без прерывания обслуживания.

Данный алгоритм является упрощенным вариантом алгоритма Ф. Корбато, который предназначался для использования в экспериментальной системе с разделением времени на базе машины ИБМ 7094. Согласно алгоритму Корбато программа, состоящая из w_p слов, поступает в очередь с номером

$$n = \left\lfloor \log_2\left(\left\lceil \frac{w_p}{w_q} \right\rceil + 1\right)\right\rfloor, \quad n = 1, 2, \ldots, N, \qquad (3.59)$$

где w_q — количество слов, передаваемое в оперативную память из внешней памяти и в обратном направлении в течение кванта времени q, а символ [ ] означает целую часть числа.

Программа из очереди с номером n получает квант обслуживания $2^n \cdot q$ и при незавершенном обслуживании переходит в очередь $(n+1)$. Если в течение времени обслуживания программы из очереди с номером n появляется программа из очереди $n' < n$, то ее обслуживание прерывается, она возвращается в начало очереди с номером n, и начинается обслуживание программы из очереди n'.

Номер очереди с наименьшим приоритетом определяется из аналитической зависимости:

$$N = \left[\log_2 \left(\left[\frac{w_{p_{max}}}{p \cdot w_q} \right] + 1 \right) \right], \qquad (3.60)$$

где p — доля времени обмена;

$w_{p_{max}}$ — наибольший допустимый объем программы в словах.

Последовательность, в которой обслуживаются различные очереди, и последовательность перехода из одной очереди в другую в алгоритме Корбато такая же, как и в циклическом алгоритме с N очередями. Имеет место отличие в обслуживании заявок очереди с номером N, а именно: программа из очереди N после получения кванта $2^N \cdot q$ поступает на дообслуживание, если оно требуется, в конец этой же очереди.

В рассмотренных выше алгоритмах просматривается идея быстрейшего завершения «коротких» программ при четко выраженном стремлении к сокращению количества обменов с внешней памятью с учетом несовершенства обычного циклического алгоритма планирования.

Смешанный алгоритм обслуживания

Рассмотрим пример использования смешанного алгоритма обслуживания, представляющего собой сочетание обычного циклического и многоприоритетного алгоритмов обслуживания. Этот алгоритм использован в системе с разделением времени на базе ЭВМ *AN/FSO*-32 фирмы ИБМ и был подобран в результате длительных экспериментальных исследований.

Из программ, находящихся в системе, организуются три очереди. Вновь поступающие программы попадают в первую очередь, где получают самое большое три цикла обслуживания по 600 мсек в соответствии с циклическим алгоритмом обслуживания. Программы, требующие дообслуживания, поступают в очередь 2, где каждая обслуживается за один цикл и получает по шести квантов по 600 мсек, после чего, если обработка ее не завершена, она поступает в очередь 3. В этой очереди программа обслуживается в соответствии с циклическим алгоритмом и может получить до 20 квантов по 600 мсек.

После окончания каждого кванта обслуживание в очередях 2 и 3 может прерваться, если за это время появится программа в очереди 1. Последовательность обслуживания очередей устанавливается в соответствии с многоприоритетным циклическим алгоритмом обслуживания.

Приоритетные дисциплины обслуживания

В реальных условиях функционирования ВС различные задачи могут существенно различаться относительной важностью и продолжительностью решения. Это обстоятельство заставляет разрабатывать алгоритм различных дисциплин приоритетного обслуживания, предоставляя преимущество наиболее важным задачам. По принципу назначения приоритетов принято различать дисциплины с фиксированными и динамическими приоритетами.

Одной из наиболее распространенных дисциплин в алгоритмах диспетчеризации вычислений является дисциплина обслуживания с фиксированными относительными приоритетами для отдельных потоков заявок. Отличительная особенность этой дисциплины состоит в том, что появление заявки более высокого приоритета не вызывает прерывания обслуживания заявок с более низкими приоритетами, если они уже обслуживаются.

Простейшей моделью дисциплины обслуживания с относительными приоритетами может служить управляющая система, в которую поступают заявки двух категорий срочности: срочные и простые. Примером такой ситуации является одновременная обработка в вычислительном центре статистических данных ежедневной и пятидневной отчетности.

Анализ рассматриваемой модели проводится при следующих предложениях:

1) заявки, имеющие две категории срочности (срочные и простые), аппроксимируются пуассоновскими потоками с параметрами λ_1 и λ_2 и не покидают систему, пока не обслужатся;

2) обслуживающая система время от времени выходит из строя также по закону Пуассона с параметром λ_0. Во время восстановления работоспособности системы новые требования не принимаются, т. е. вход системы перекрывается и открывается только в момент восстановления работоспособности. Предполагается, что выход системы из строя одинаково возможен как в период обслуживания заявки, так и в период, когда она свободна. Если система вышла из строя в период обслуживания простой заявки, то после восстановления начинает обслуживаться срочная заявка, если таковая имеется;

3) время восстановления системы и время обслуживания срочных и простых заявок подчиняется показательному закону распределения с параметрами μ_0, μ_1, μ_2;

4) дисциплина очереди такова, что срочные заявки обслуживаются всегда раньше простых независимо от времени их поступления.

Если на обслуживании находится простая заявка и в систему поступает срочная, то обслуживание не прерывается. В пределах одной категории срочности заявки обслуживаются в порядке их поступления.

К основным характеристикам функционирования такой системы относятся:

математическое ожидание и дисперсия числа срочных и простых заявок в системе;

математическое ожидание длины очереди простых и срочных заявок;

среднее время пребывания в системе и ожидания в очереди для заявок каждой категории срочности.

Рассматриваемая система обслуживания описывается следующими уравнениями:

$$
\left.
\begin{gathered}
P'_{00}(t) = -(\lambda_0 + \lambda_1 + \lambda_2) P_{00}(t) + \mu_1 P_{01}(t) + \mu_2 Q_{10}(t) + \\
+ \mu_0 H_{00}(t) \\
P'_{0K}(t) = -(\lambda_0 + \lambda_1 + \lambda_2 + \mu_1) P_{0K}(t) + \lambda_1 P_{0,\,K-1}(t) + \\
+ \mu_1 P_{0,\,K+1}(t) + \mu_2 Q_{1K}(t) + \mu_0 H_{0K}(t) \\
P'_{i1}(t) = -(\lambda_0 + \lambda_1 + \lambda_2 + \mu_1) P_{i1}(t) + \lambda_2 P_{i-1,\,1}(t) + \\
+ \mu_1 P_{i,\,2}(t) + \mu_2 Q_{i+1,\,1}(t) + \mu_0 H_{i1}(t) \\
P'_{i,\,K+1}(t) = -(\lambda_0 + \lambda_1 + \lambda_2 + \mu_1) P_{i,\,K+1}(t) + \lambda_1 P_{iK}(t) + \\
+ \lambda_2 P_{i-1,\,K+1}(t) + \mu_1 P_{i,\,K+2}(t) + \mu_2 Q_{i+1,\,K+1}(t) + \\
+ \mu_0 H_{i,\,K+1}(t) \\
Q'_{i0}(t) = -(\lambda_0 + \lambda_1 + \lambda_2 + \mu_2) Q_{i0}(t) + \lambda_2 Q_{i-1,\,0}(t) + \\
+ \mu_1 P_{i,\,1}(t) + \mu_2 Q_{i+1,\,0}(t) + \mu_0 H_{i0}(t) \\
Q'_{iK}(t) = -(\lambda_0 + \lambda_1 + \lambda_2 + \mu_2) Q_{1K}(t) + \lambda_1 Q_{1,\,K-1}(t) \\
Q'_{i+1,\,K}(t) = -(\lambda_0 + \lambda_1 + \lambda_2 + \mu_2) Q_{i+1,\,K}(t) + \\
+ \lambda_1 Q_{i+1,\,K-1}(t) + \lambda_2 Q_{iK}(t) \\
H'_{00}(t) = -\mu_0 H_{00}(t) + \lambda_0 P_{00}(t) \\
H'_{0K}(t) = -\mu_0 H_{0K}(t) + \lambda_0 P_{0K}(t) \\
H'_{i0}(t) = -\mu_0 H_{i0}(t) + \lambda_0 Q_{i0}(t) \\
H'_{iK}(t) = -\mu_0 H_{iK}(t) + \lambda_0 P_{iK}(t) + \lambda_0 Q_{iK}(t),\ i,\ K = \\
= 1,\ 2,\ 3,\ \ldots
\end{gathered}
\right\} \qquad (3.61)
$$

Здесь $P_{00}(t)$ — вероятность того, что в момент t в системе отсутствуют заявки, и система исправна;

$H_{00}(t)$ — вероятность того, что в момент t в системе отсутствуют заявки, и она восстанавливается;

$P_{iK}(t)$ — вероятность того, что в системе в момент t находится K срочных и i простых заявок, система исправна, и обслуживается очередная срочная заявка;

$Q_{iK}(t)$ — вероятность того, что в системе в момент t находится K срочных и i простых заявок, система ис-

правна, и обслуживается очередная простая заявка;

$H_{iK}(t)$ — вероятность того, что в системе в момент t находится K срочных и i простых заявок и выполняется ее восстановление.

Система (3.61) для случая исследования стационарного режима может быть сведена к следующей системе линейных алгебраических уравнений:

$$\left.\begin{gathered}
(\lambda_0+\lambda_1+\lambda_2)P_{00}=\mu_1P_{01}+\mu_2q_{10}+\mu_0h_{00}\\
(\lambda_0+\lambda_1+\lambda_2+\mu_1)P_{0K}=\lambda_1P_{0,\,K-1}+\mu_1P_{0,\,K+1}+\mu_2q_{1K}+\\
+\mu_0h_{0K}\\
(\lambda_0+\lambda_1+\lambda_2+\mu_1)P_{i1}=\lambda_2P_{i-1,\,1}+\mu_1P_{i,\,2}+\mu_2q_{i+1,\,1}+\\
+\mu_0h_{i1}\\
(\lambda_0+\lambda_1+\lambda_2+\mu_1)P_{i,\,K+1}=\lambda_1P_{iK}+\lambda_2P_{i-1,\,K+1}+\\
+\mu_1P_{i,\,K+2}+\mu_2q_{i+1,\,K+1}+\mu_0h_{i,\,K+1}\\
(\lambda_0+\lambda_1+\lambda_2+\mu_2)q_{i0}=\lambda_2q_{i-1,\,0}+\mu_1P_{i1}+\mu_2q_{i+1,\,0}+\mu_0h_{i0}\\
(\lambda_0+\lambda_1+\lambda_2+\mu_2)q_{iK}=\lambda_1q_{i,\,K-1}\\
(\lambda_0+\lambda_1+\lambda_2+\mu_2)q_{i+1,\,K}=\lambda_1q_{i+1,\,K-1}+\lambda_2q_{iK}\\
\mu_0h_{00}=\lambda_0P_{00}\\
\mu_0h_{0K}=\lambda_0P_{0K}\\
\mu_0h_{i0}=\lambda_0q_{i0}\\
\mu_0h_{iK}=\lambda_0P_{iK}+\lambda_0q_{iK}
\end{gathered}\right\}\qquad(3.62)$$

Решаем эту систему уравнений методом производящих функций. Так как обслуживающая система может находиться в одном из возможных состояний, т. е. занята обслуживанием срочной или простой заявки, восстановлением или свободна от того и другого, то сумма вероятностей этих состояний равна единице. Исходя из этого введем производящую функцию

$$F(x,\ y)=F_1(x,\ y)+F_2(x,\ y)+F_3(x,\ y)+p_{00},\qquad(3.63)$$

где

$$F_1(x,\ y)=\sum_{i=0}^{\infty}\sum_{k=1}^{\infty}p_{ik}x^iy^k;$$

$$F_2(x,\ y)=\sum_{k=0}^{\infty}\sum_{i=1}^{\infty}q_{ik}x^iy^k;$$

$$F_3(x,\ y)=\sum_{i=0}^{\infty}\sum_{k=0}^{\infty}h_{ik}x^iy^k.$$

Данные функции являются частичными производящими функциями, определенными при $|x|\leqslant 1$ и $|y|\leqslant 1$. Функция F_1 (1,1) есть

вероятность того, что система занята обслуживанием срочной заявки, $F_2(1,1)$ — простой и $F_3(1,1)$ — восстановлением. Кроме того, $F(1,1)=1$.

Производящую функцию и ее составляющие находим следующим образом. Умножив уравнения системы (3.62) на $x^i y^k$ и просуммировав по i и k, получим зависимости между частичными производящими функциями, из которых находим:

$$F_1(x, y)=-\frac{(\lambda_0+\lambda_1+\lambda_2+\mu_2-\lambda_2 x)\left(\lambda_1+\lambda_2+\mu_2-\lambda_1 y-\lambda_2 x-\frac{\mu_2}{x}\right)\cdot R_0(x)}{(\lambda_0+\lambda_1+\lambda_2+\mu_2-\lambda_1 y-\lambda_2 x)\left(\lambda_1+\lambda_2+\mu_1-\lambda_1 y-\lambda_2 x-\frac{\mu_1}{y}\right)}-$$

$$-\frac{\lambda_1+\lambda_2-\lambda_1 y-\lambda_2 x}{\lambda_1+\lambda_2+\mu_1-\lambda_1 y-\lambda_2 x-\frac{\mu_1}{y}}\cdot p_{00}; \qquad (3.64)$$

$$F_2(x, y)=\frac{\lambda_0+\lambda_1+\lambda_2+\mu_2-\lambda_2 x}{\lambda_0+\lambda_1+\lambda_2+\mu_2-\lambda_1 y-\lambda_2 x}\cdot R_0(x); \qquad (3.65)$$

$$F_3(x, y)=\rho_0\frac{A(x, y) R_0(x)-B(x, y) p_{00}}{(\lambda_0+\lambda_1+\lambda_2+\mu_2-\lambda_1 y-\lambda_2 x)\left(\lambda_1+\lambda_2+\mu_1-\lambda_1 y-\lambda_2 x-\frac{\mu_1}{x}\right)}, \qquad (3.66)$$

где

$$A(x, y)=(\lambda_0+\lambda_1+\lambda_2+\mu_2-\lambda_2 x)\left[\mu_2\left(\frac{1}{x}-1\right)-\mu_1\left(\frac{1}{y}-1\right)\right];$$

$$B(x, y)=(\lambda_0+\lambda_1+\lambda_2+\mu_2-\lambda_1 y)\,\mu_1\left(\frac{1}{y}-1\right);$$

$$R_0(x)=\sum_{i=1}^{\infty} q_{i0} x^i.$$

Выполнив подстановку (3.64) — (3.66) в (3.63), найдем производящую функцию

$$F(x, y)=\frac{(1+\rho_0)\,[A(x, y) R_0(x)-B(x, y) P_{00}]}{(\lambda_0+\lambda_1+\lambda_2+\mu_2-\lambda_1 y-\lambda_2 x)\left(\lambda_1+\lambda_2+\mu_1-\lambda_1 y-\lambda_2 x-\frac{\mu_1}{y}\right)}. \qquad (3.67)$$

Для определения неизвестной $R_0(x)$ поступим следующим образом. Так как $F(x, y)$ определена при $|x|\leqslant 1$ и $|y|\leqslant 1$, то нули знаменателя выражения (3.67) должны совпадать с соответствующими нулями числителя. Приравняв знаменатель в (3.67) к нулю и выполнив замену $y=\frac{1}{v}$, получим следующее квадратное уравнение:

$$\mu_1 v^2-(\mu_1+\lambda_1+\lambda_2-\lambda_2 x)\,v+\lambda_1=0, \qquad (3.68)$$

так как при $|x|\leqslant 1$ и $|y|\leqslant 1$ сомножитель

$$\lambda_0+\lambda_1+\lambda_2+\mu_2-\lambda_1 y-\lambda_2 x\neq 0.$$

Решив квадратное уравнение (3.68), найдем его корни:

$$v_{1,2}(x) = \frac{1}{2\mu_1}(\mu_1 + \lambda_1 + \lambda_2 - \lambda_2 x \pm$$

$$\pm \sqrt{(\mu_1 + \lambda_1 + \lambda_2 - \lambda_2 x)^2 - 4\mu_1\lambda_1}. \qquad (3.69)$$

Заметим, что $v_2(x)$ — величина, обратная нулю знаменателя (3.67), лежащего вне единичного круга $|y| = 1$, а $v_1(x)$ — величина, обратная другому нулю знаменателя. Кроме того, на основании свойства корней квадратного уравнения

$$\mu_1 + \lambda_1 + \lambda_2 - \lambda_2 x = \mu_1(v_1 + v_2) \text{ и } \lambda_1 = \mu_1 v_1 v_2.$$

Приравняв числитель в (3.67) к нулю, сделав замену $y = \frac{1}{v_2}$, выполнив некоторые упрощения, основанные на особенностях корней v_1 и v_2, умножив числитель и знаменатель полученного выражения на $(1 - v_1)$, окончательно найдем

$$R_0(x) = \frac{(\lambda_0 + \mu_2 - \mu_1(1 - v_2))\rho_2 x}{(\lambda_0 + \lambda_1 + \lambda_2 + \mu_2 - \lambda_2 x)(1 - \rho_2 x - v_1)} \cdot p_{00}, \qquad (3.70)$$

где

$$\rho_2 = \frac{\lambda_2}{\mu_2}.$$

Заметим, что R_0 (1) — вероятность того, что в системе отсутствуют срочные заявки при наличии хотя бы одной простой заявки и работоспособной системы. Из (3.70) или (3.67), переходя к пределу сначала при $y \to 1$, а затем при $x \to 1$, находим

$$R_0(x) = \frac{\rho_2}{1 + \rho_0} \cdot \frac{\lambda_0 + \mu_2}{\lambda_0 + \lambda_1 + \mu_2}. \qquad (3.71)$$

Применив порядок перехода к пределу в (3.67), найдем

$$p_{00} = \frac{1 - \rho_1 - \rho_2}{1 + \rho_0} = k_r(1 - \rho_1 - \rho_2), \qquad (3.72)$$

где $k_r = 1/(1 + \rho_0)$ — коэффициент готовности системы.

Из (3.64) при $x \to 1$ и $y \to 1$ с подстановкой (3.71) и (3.72) находим вероятность того, что система занята обслуживанием срочной заявки:

$$F_1(1, 1) = \frac{\rho_1}{1 + \rho_0} = \rho_1 k_r, \qquad (3.73)$$

т. е. вероятность того, что система занята обслуживанием срочной заявки, равна произведению коэффициента загрузки системы на вероятность того, что система исправна.

Аналогично из (3,65) находим вероятность того, что система занята обслуживанием простой заявки:

$$F_2(1, 1) = \frac{\rho_2}{1 + \rho_0} = \rho_2 k_r, \qquad (3.74)$$

а из (3.66) — вероятность того, что система восстанавливается:

$$F_в(1, 1) = \frac{\rho_0}{1+\rho_0} = \rho_0 k_г. \qquad (3.75)$$

Математическое ожидание числа срочных заявок в системе определяется так:

$$M_D = \left.\frac{dF(x, y)}{dy}\right|_{\substack{x=1\\y=1}} = \sum_{i=0,}^{\infty}\sum_{k=1}^{\infty} k(p_{ik} + h_{ik}) + \sum_{i=1}^{\infty}\sum_{k=1}^{\infty} kq_{ik}. \qquad (3.76)$$

Второй начальный момент M_p^* находим из выражения для второй частной производной функции по y:

$$\left.\frac{d^2F(x, y)}{dy^2}\right|_{\substack{x=1\\y=1}} = \sum_{i=0}^{\infty}\sum_{k=1}^{\infty} k(k-1)(p_{ik}+h_{ik}) +$$

$$+ \sum_{i=1}^{\infty}\sum_{k=1}^{\infty} k(k-1)q_{ik} = M^*{}_p - M_D, \qquad (3.77)$$

а дисперсию — из выражения

$$D_p = M^*{}_p - M_D{}^2. \qquad (3.78)$$

В связи с тем, что для срочных заявок дифференцирование производящей функции производится по y, то (3.67) можно записать в упрощенном виде:

$$F(1, y) = \frac{1+\rho_0}{1-\rho_1 y} \cdot \left(\frac{\rho_2}{1+\rho_0} \cdot \frac{\lambda_0+\mu_2}{\lambda_0+\lambda_1+\mu_2-\lambda_1 y} + p_{00}\right). \qquad (3.79)$$

Взяв производную по y от (3.79) при $y \to 1$, после упрощения имеем

$$M_p = \frac{\rho_1}{1-\rho_1}\left(1 + \frac{\alpha_1\rho_2}{1+\alpha_0\rho_0}\right), \qquad (3.80)$$

где

$$\alpha_0 = \mu_0/\mu_2, \quad \alpha_1 = \mu_1/\mu_2.$$

Математическое ожидание числа срочных заявок в очереди определим как разность между математическим ожиданием числа срочных заявок в системе и на обслуживании, т. е. разность между (3.80) и (3.73):

$$L_p = \frac{\rho_1}{1-\rho_1}\left(\frac{\rho_0+\rho_1}{1+\rho_0} + \frac{\alpha_1\rho_2}{1+\alpha_0\rho_0}\right). \qquad (3.81)$$

Среднее время пребывания срочной заявки в системе будет определяться по формуле

$$W_p = \frac{1}{\mu_1 k_г(1-\rho_1)} \cdot \left(1 + \frac{\alpha_1\rho_2}{1+\alpha_0\rho_0}\right), \qquad (3.82)$$

а среднее время ожидания в очереди:

$$W^*{}_p = \frac{1}{\mu_1 k_г(1-\rho_1)} \cdot \left(\frac{\rho_0+\rho_1}{1+\rho_0} + \frac{\alpha_1\rho_2}{1+\alpha_0\rho_0}\right), \qquad (3.83)$$

так как интенсивность поступивших заявок будет равна $\lambda_1 k_г$.

Второй начальный момент и дисперсию для срочных заявок найдем, используя зависимости (3.77) и (3.78). Взяв вторую производную по y от (3.79) при $y \to 1$ и подставив полученное выражение и выражение (3.80) в (3.77), найдем второй начальный момент:

$$M^*_p = \frac{\rho_1}{(1-\rho_1)^2}\left[(1+\rho_1)\left(1+\frac{\alpha_1\rho_2}{1+\alpha_0\rho_0}\right)+\frac{2\alpha_1^2\rho_1\rho_2(1-\rho_1)}{(1+\alpha_0\rho_0)^2}\right]. \quad (3.84)$$

Выполнив подстановку (3.80) и (3.84) в (3.78), найдем дисперсию

$$D_p = \frac{\rho_1}{(1-\rho_1)^2}\left[1+\frac{\alpha_1\rho_2(1-\rho_1)}{1+\alpha_0\rho_0}+\alpha_1^2\rho_1\rho_2\frac{2(1-\rho_1)-\rho_2}{(1+\alpha_0\rho_0)^2}\right]. \quad (3.85)$$

Вероятность того, что в системе находится ровно k ($k=0, 1, 2, \ldots$) срочных заявок, определим по формуле

$$p_k = \frac{1}{k!}\left.\frac{\partial^k F(1, y)}{\partial y^k}\right|_{y=0} =$$

$$= \rho_1^k(1+\rho_0)\left\{p_{00}+R_0(1)\frac{\lambda_0+\lambda_1+\mu_2}{\mu_1-\lambda_0-\lambda_1-\mu_2}\left[\left(\frac{\mu_1}{\lambda_0+\lambda_1+\mu_2}\right)^{k+1}-1\right]\right\}. \quad (3.86)$$

Вероятность того, что в системе находится ровно k ($k=1, 2, 3, \ldots$) срочных заявок и система занята обслуживанием срочной заявки:

$$p^*_k = \frac{1}{k!}\left.\frac{\partial^k F_1(1, y)}{\partial y^k}\right|_{y=0} =$$

$$= \rho_1^k\left\{p_{00}+R_0(1)\frac{\lambda_0+\lambda_1+\mu_2}{\mu_1-\lambda_0-\lambda_1-\mu_2}\left[\left(\frac{\mu_1}{\lambda_0+\lambda_1+\mu_2}\right)^{k}-1\right]\right\}. \quad (3.87)$$

Вероятность того, что в системе находится ровно k ($k=0, 1, 2, \ldots$) срочных заявок и система занята обслуживанием простой заявки:

$$R_k = \frac{1}{k!}\left.\frac{\partial^k F_2(1, y)}{\partial y^k}\right|_{y=0} = \frac{\lambda_1^k}{(\lambda_0+\lambda_1+\mu_2)^k}\cdot R_0(1). \quad (3.88)$$

Вероятность того, что в системе находится ровно k ($k=0, 1, 2, \ldots$) срочных заявок и система восстанавливается:

$$H_k = \frac{1}{k!}\left.\frac{\partial^k F_3(1, y)}{\partial y^k}\right|_{y=0} =$$

$$= \rho_0\rho_1^k\left\{p_{00}+R_0(1)\frac{\lambda_0+\lambda_1+\mu_2}{\mu_1-\lambda_0-\lambda_1-\mu_2}\left[\left(\frac{\mu_1}{\lambda_0+\lambda_1+\mu_2}\right)^{k+1}-1\right]\right\}. \quad (3.89)$$

Аналогичным образом можно определить соответствующие характеристики для простых заявок. Производящую функцию (3.67) перепишем в упрощенном виде:

$$F(x, 1) = \frac{\lambda_0+\mu_2-\mu_1(1-v_2)}{\lambda_0+\lambda_2+\mu_2-\lambda_2 x}\cdot\frac{1+\rho_0}{1-\rho_2 x-v_1}p_{00}. \quad (3.90)$$

Взяв производную по x от (3.90) при $x \to 1$ и воспользовавшись тем, что из (3,69)

$$v_1(1) = \rho_1;\ v_2(1) = 1;$$

$$v_1'(1) = \frac{\lambda_2 \rho_1}{\mu_1 - \lambda_1} \text{ и } v_2'(1) = -\frac{\lambda_2}{\mu_1 - \lambda_1},$$

найдем математическое ожидание числа простых заявок в системе:

$$M_q = \frac{\rho_2}{1-\rho_1}\left[\frac{\rho_1 + \alpha_1(1-\rho_1)}{\alpha_1(1-\rho_1-\rho_2)} - \frac{\rho_1}{1+\alpha_0\rho_0}\right]. \tag{3.91}$$

Математическое ожидание числа простых заявок в очереди равно:

$$L_q = \frac{\rho_2}{1-\rho_1}\left[\frac{\rho_1 + \alpha_1(1-\rho_1)}{\alpha_1(1-\rho_1-\rho_2)} - \frac{\rho_1}{1+\alpha_0\rho_0} - \frac{1-\rho_1}{1+\rho_0}\right]. \tag{3.92}$$

Среднее время пребывания простой заявки в системе:

$$W_q = \frac{1}{\mu_1 k_\Gamma (1-\rho_1)}\left[\frac{\rho_1 + \alpha_1(1-\rho_1)}{1-\rho_1-\rho_2} - \frac{\alpha_1\rho_1}{1+\alpha_0\rho_0}\right]. \tag{3.93}$$

Среднее время ожидания в очереди:

$$W^*_q = \frac{1}{\mu_2 k_\Gamma (1-\rho_1)}\left[\frac{\rho_1 + \alpha_1(1-\rho_1)}{\alpha_1(1-\rho_1-\rho_2)} - \frac{\rho_1}{1+\alpha_0\rho_0} - \frac{1-\rho_1}{1+\rho_0}\right]. \tag{3.94}$$

Взяв вторую производную по x от (3.90) при $x \to 1$ и подставив в полученное выражение

$$v''_{1,2}(1) = \pm \frac{2\rho_1\rho_2^2}{\alpha_1^2(1-\rho_1)^3},$$

найдем, как и для срочных заявок, второй начальный момент для простых заявок:

$$M^*_q = \frac{\rho_2}{(1-\rho_1)^2}\left\{\left[\frac{\rho_1 + \alpha_1(1-\rho_1)}{\alpha_1(1-\rho_1-\rho_2)} - \frac{\rho_1}{1+\alpha_0\rho_0}\right]\left[2\rho_2\frac{\rho_1 + \alpha_1(1-\rho_1)}{\alpha_1(1-\rho_1-\rho_2)} + (1-\rho_1)\right] + \frac{2\rho_1\rho_2}{\alpha_1(1-\rho_1)}\left[\frac{1}{\alpha_1(1-\rho_1-\rho_2)} - \frac{1}{1+\alpha_0\rho_0}\right] - \frac{2\rho_1\rho_2(1-\rho_1)}{(1+\alpha_0\rho_0)^2}\right\} \tag{3.95}$$

и дисперсию

$$D_q = \frac{\rho_2}{(1-\rho_1)^2}\left\{\left[\frac{\rho_1 + \alpha_1(1-\rho_1)}{\alpha_1(1-\rho_1-\rho_2)} - \frac{\rho_1}{1+\alpha_0\rho_0}\right]\left[1-\rho_1\left(1-\frac{\rho_2}{1+\alpha_0\rho_0}\right) + \rho_2\frac{\rho_1 + \alpha_1(1-\rho_1)}{\alpha_1(1-\rho_1-\rho_2)}\right] + \frac{2\rho_1\rho_2}{\alpha_1(1-\rho_1)}\left[\frac{1}{\alpha_1(1-\rho_1-\rho_2)} - \frac{1}{1+\alpha_0\rho_0} - \frac{\alpha_1(1-\rho_1)^2}{(1-\alpha_0\rho_0)^2}\right]\right\}. \tag{3.96}$$

Зная среднее число срочных (3.81) и простых (3.82) заявок в очереди, найдем среднюю длину общей очереди:

$$L = L_p + L_q = \frac{1}{1-\rho_1}\left[\rho_2 \frac{\rho_1 + \alpha_1(1-\rho_1)}{\alpha_1(1-\rho_1-\rho_2)} + \right.$$
$$\left. + \frac{\rho_1(\rho_0+\rho_1+\rho_2)-\rho_2}{1+\rho_0} - \frac{\rho_1\rho_2(1-\alpha_1)}{1+\alpha_0\rho_0}\right]. \quad (3.97)$$

Среднее число заявок в системе независимо от приоритета находим, суммируя (3.80) и (3.91):

$$M = M_p + M_q = \frac{1}{1-\rho_1}\left[\rho_1 + \rho_2 \frac{\rho_1 + \alpha_1(1-\rho_1)}{\alpha_1(1-\rho_1-\rho_2)} - \frac{\rho_1\rho_2(1-\alpha_1)}{1+\alpha_0\rho_0}\right]. \quad (3.98)$$

Таким образом, для случая двух входящих потоков (простые и срочные заявки) получены основные характеристики дисциплины обслуживания с фиксированными относительными приоритетами.

В алгоритмах диспетчеризации с относительными приоритетами обслуживание каждой вновь поступившей заявки может быть начато только по окончании начатого ранее обслуживания предыдущей заявки, даже если последняя имеет более низкий приоритет. Вследствие этого ограничения длительность ожидания в очереди даже для заявок высокого приоритета может оказаться весьма значительной.

Уменьшение времени ожидания заявок с высоким приоритетом может быть достигнуто введением так называемых абсолютных приоритетов в обслуживании.

Обслуживание заявок с низкими приоритетами прерывается всякий раз, когда в систему поступают заявки более высокого приоритета. Заявки с равными приоритетами обслуживаются в порядке их поступления в систему.

Длительность ожидания в очереди заявок с низкими приоритетами при наличии прерываний зависит также от принятого способа возобновления прерванного обслуживания. Характерными являются дисциплины с сохранением прерванного кванта обслуживания (с возобновлением прерванного обслуживания) и с потерей прерванного кванта обслуживания. При использовании алгоритмов диспетчеризации с абсолютными приоритетами возникает проблема выбора приемлемого решения на изменение времени задержки для заявок низшего приоритета с выигрышем во времени ожидания заявок высшего приоритета. При оценке эффективности применения абсолютных приоритетов необходимо определять изменение общих потерь с учетом штрафа за ожидание и интенсивности потоков заявок каждого приоритета. Значительные расходы времени и памяти для запоминания текущего состояния при прерываниях и последующих восстановлениях обслуживания заявок с абсолютными приоритетами сдерживают практические применения этой дисциплины обслуживания.

Перечисленные выше трудности практического использования дисциплины с абсолютными приоритетами натолкнули на необходимость поиска некоторой промежуточной дисциплины между абсолютными и относительными приоритетами (адаптивное обслуживание). При этой дисциплине обслуживания определяется целесооб-

разность прерывания обслуживания заявок с низшими приоритетами при появлении заявок с высшими приоритетами. Если заявка почти полностью обслужена, то может оказаться выгодным не прерывать обслуживания, а довести его до конца, не увеличивая системное время.

В литературе можно встретить критерий необходимости завершения обслуживания заявки с низким приоритетом в виде следующего неравенства:

$$\Delta T_{ji} < \frac{T_i}{\alpha_i} \alpha_j,$$

где ΔT_{ji} — время, необходимое для завершения обслуживания заявки j-го типа в момент поступления заявки более высокого i-го приоритета;

α_i, α_j — штраф за единицу времени ожидания i-й и j-й заявок соответственно;

T_i — время обслуживания поступившей заявки i-го типа.

Дисциплины с динамическими приоритетами

При рассмотрении дисциплины диспетчеризации с фиксированными приоритетами предполагалась неизменной приоритетность заявок в течение времени ожидания и обработки их системой. В силу того что относительная важность заявок может измениться в зависимости от времени задержки обслуживания и изменения времени обслуживания, возникает необходимость в таких приоритетных дисциплинах обслуживания, приоритетность заявок в которых зависит от реального времени обслуживания или от длительности ожидания заявок в памяти машин.

В тех случаях, когда интервал времени, в течение которого эти параметры можно считать постоянными, много больше средней продолжительности обслуживания заявок, настройка алгоритма диспетчеризации на изменяющиеся параметры входных потоков заявок может быть реализована перестройкой алгоритмов диспетчеризации.

При быстром изменении параметров входных потоков перестройка алгоритмов диспетчеризации оказывается малоэффективной как вследствие ее инерционности, так и вследствие увеличения затрат времени на изменение режима диспетчеризации. Здесь оправдано применение алгоритмов диспетчеризации с динамическими приоритетами, обладающих большей способностью настройки на изменяющиеся характеристики, но несколько более сложных в реализации.

При проектировании алгоритма диспетчеризации вычислений в ряде случаев оказывается целесообразным использовать сочетание нескольких дисциплин обслуживания, позволяющее получить рациональный компромисс между достоинствами и затратами на создание каждой из них.

Для диспетчеризации обслуживания однотипных заявок или заявок, поступающих от одного источника (что характерно для работы

различных модулей операционной системы), наиболее целесообразно использовать дисциплину обслуживания в порядке поступления заявок.

Планирование загрузки ВС в многопрограммном режиме

Одной из основных проблем многопрограммной работы ВС является нахождение оптимального критерия плана реализации заданного набора задач. Основная цель планирования при многопрограммной работе состоит в решении задач оптимального распределения квантов времени при реализации задач на одном процессоре и, кроме того, для мультипроцессорной ВС — распределение задач между машинами системы.

Задача определения оптимального порядка обработки программных блоков на ВС известна в литературе в различных постановках и представляет собой серьезную самостоятельную проблему, называемую в исследовании операций задачей упорядочения. Содержанием задачи упорядочения является выбор дисциплины обслуживания с таким расчетом, чтобы принятый критерий качества функционирования достигал экстремального значения.

Вычислительную систему, выбор работ для которой производится из некоторого количества имеющихся заявок, можно рассматривать как систему массового обслуживания поступающих заявок на выполнение вычислительных работ. В этом случае в качестве критерия эффективности можно взять общее время обслуживания T_0 в многопрограммном режиме заданной группы заявок с его последующей минимизацией. Эта задача может решаться с учетом приоритетов заявок при условии выполнения заданных сроков реализации работ с функциями «премий» и «штрафов» за соответствующие отклонения. Попытка решения подобной задачи методом перебора приводит к необходимости реализации большого объема вычислений. В силу практической важности подобных задач в последнее время появилось значительное число вариантов решений, позволяющих избежать простого перебора.

Рассмотрим один из вариантов решения для следующей задачи. Известно множество работ

$$P\{P_1,\ P_2,\ \ldots,\ P_l\},$$

которое необходимо выполнить на одной машине. Любая работа P_i реализуется за τ_i единиц машинного времени. Предполагается, что машина работает непрерывно за исключением времени настройки с работы на работу. Задана матрица настройки $|\tau_{ij}|$ $(i, j=1, 2, \ldots l)$, где τ_{ij} — необходимое время настройки при переходе от работы i к работе j.

Заданы также некоторые функции штрафа $J_{is}(t_{is})$, определяющие размер штрафа за выполнение работы is в момент ts. С помощью последовательности целых чисел $(i_1, i_2, \ldots, i_l)$ от 1 до l представляется последовательность выполняемых работ. При этом

порядке время завершения работы с индексом is будет определяться зависимостью:

$$t_{is} = \tau_{0j_1} + \tau_{j_1 j_2} + \ldots + \tau_{i_{s-1} j_s} + \sum_{\lambda=1}^{s} \tau_{j\lambda}.$$

Тогда суммарные штрафы выполнения работ в принятой последовательности будут равны

$$\sum_{s=1}^{l} J_{is}(t_{is}).$$

Требуется отыскать последовательность $(i_1, i_2, \ldots, i_l)$, минимизирующую суммарную функцию штрафа. Решение задачи выполняется методами динамического программирования.

При числе процессоров $n>1$ задача планирования загрузки существенно усложняется. В этом случае перед решением задачи упорядочения необходимо выполнить распределение программ (программных блоков) между процессорами. В задаче распределения требуется сопоставить каждой вершине граф-схемы алгоритма (оператору программы) некоторый процессор и определить последовательность реализации операторов таким образом, чтобы либо минимизировать общее время решения данной задачи, либо получить максимальную загрузку ВС при обеспечении заданной надежности. При решении задачи распределения необходимо соблюдать требование очередности выполнения последовательности операторов в соответствии с алгоритмом решения задачи. Этой особенностью задача распределения отличается от задачи оптимального распределения m независимых работ между n исполнителями.

Более строгая постановка задачи планирования загрузки мультипроцессорной ВС может быть получена при использовании функции потерь, аналогичной по своему смысловому содержанию функции суммарных штрафов.

Смысловое содержание понятия потерь, характеризующих отклонения от оптимальной по отношению к определенному критерию организации работы ВС, может быть различным. Потери характерны для любой ВС независимо от ее назначения. Для универсальных ВС это могут быть потери из-за неполной загрузки системы. В то же время даже при полной загрузке, когда несколько задач решаются одновременно, потери могут возникать из-за некачественной дисциплины обслуживания (более важные задачи решаются относительно медленно). Для ВС, работающей в реальном времени, потери возникают, если в режиме перегрузки часть входных данных не успевает обрабатываться, а также при отказах в одном из процессоров или внешних устройствах.

Определенные трудности имеют место при формировании функции потерь на основе составляющих факторов в силу их различных весов. Один из принятых в исследовании операций подходов предполагает формирование ее в виде линейной функции составляющих факторов, равной сумме произведений относительных значений этих

факторов на их вес, т. е. долю их вклада в общую величину потерь. Весовые характеристики отдельных факторов обычно определяются одним из методов экспертных оценок. В этом случае задача планирования загрузки предполагает минимизацию общей стоимости потерь, связанных с реализацией исходного алгоритма при заданной структуре ВС.

Для мультипроцессорной ВС зададим распределение операторов (вершин граф-схемы) блок-схемы алгоритма между процессорами матрицей распределения вершин $|x_n|$. Элемент матрицы x_{nj_1} равен единице, если оператор счета, соответствующий вершине с номером j_1, обрабатывается на n-м процессоре. Последовательность обработки операторов на каждом из процессоров характеризуется матрицей порядка $|z_n|$, в которой элемент $z_{n_{j1},\ i1}$ равен единице, если оператор счета, соответствующий вершине с номером j_1, обрабатывается на n-м процессоре i_1-м по порядку номером. Структура связей в мультипроцессорной ВС характеризуется матрицей связи процессоров $|y|$ размерностью $n \times n$. Структуру ВС предстоит выбрать из некоторого множества структур, т. е. необходимо выбрать количество процессоров и схему их объединения.

В этой постановке задача планирования загрузки мультипроцессорной ВС может быть сформулирована так. Пусть дано описание некоторой задачи (класса задач) графом G и набором процессоров, каждый из которых обладает множеством операций S с их временными характеристиками. Требуется определить структуру системы, характеризуемую матрицей связи процессоров $|y^*|$, и преобразование графа G в G^* с заданным параллелизмом, а также матрицы распределения вершин $|x^*|$ и порядка $|z^*|$ таким образом, чтобы общая стоимость потерь L, связанных с реализацией исходного алгоритма, минимизировалась:

$$L = \min_{n,\ i,\ q} (L_{n,\ i,\ q})$$

при условии, что общий объем оборудования y^* был бы не больше заданного y_3. Выбор производится среди всевозможных комбинаций значений n, i, q, характеризующих варианты структуры системы, распределение операторов счета по процессорам и порядок их обработки.

На основе комплексной задачи планирования загрузки и структуры мультипроцессорной системы можно выделить следующие варианты задач:

1. В случае фиксированных матриц $|y^*|$ и $|x^*|$ основной проблемой является определение оптимальной последовательности отдельных задач или их частей, предназначенных для решения на данном процессоре системы. Типичным примером этого случая является задача мультипрограммирования.

2. Когда задана только матрица $|y^*|$, т. е. фиксирована структура системы, основными являются проблемы оптимального распре-

деления вершин по процессорам и выбора оптимальной последовательности реализации этих вершин.

3. Если число процессоров системы переменно, а максимальное время решения задач фиксировано, то проблема сводится к определению оптимальной последовательности обработки для каждого из процессоров.

4. При переменном числе процессоров системы и заданных ограничениях по надежности функционирования и времени реализации проблема сводится к определению оптимальной последовательности обработки операторов для каждого из процессоров при соответствующем дублировании.

5. Если структурная матрица $|y^*|$ является полностью или частично неопределенной и допустимы структурные изменения системы с помощью дополнительного оборудования, то проблема является еще более общей и состоит в выборе числа одновременно работающих процессоров системы и определении оптимальной последовательности операторов, реализуемых на них так, чтобы удовлетворялся критерий минимальных потерь.

В конкретных случаях проблема планирования загрузки ВС ставится по-разному в зависимости от вида решаемых задач и характеристик системы. Распределение может диктоваться самой структурой алгоритма, включающего группу функционально самостоятельных, не связанных или слабо связанных задач. Весьма существенным является учет надежности процессоров системы, что в значительной мере определяет порядок распределения нагрузки.

Среди множества методов планирования загрузки в многопрограммном режиме можно выделить две разновидности; методы, предполагающие распределение функционально независимых задач, и методы распараллеливания алгоритмов и распределения ветвей по процессорам ВС.

Применительно к первой группе методов рассмотрим ряд алгоритмов оптимального распределения программ на параллельно работающие вычислительные машины. Задача ставится следующим образом.

Пусть имеется m независимых задач и n параллельно работающих машин различной производительности ($n=m$). При любом распределении время работы машины с номером j определяется соотношением

$$T_j = b_{1j}\tau_{1j} + b_{2j}\tau_{2j} + \ldots + b_{mj}\tau_{mj},$$

где τ_{ij} — время выполнения задачи i на машине j;

b_{ij} — элемент булевой матрицы распределения задач по машинам.

В качестве линейной формы, подлежащей минимизации при решении задач планирования, предполагается наибольшее время загрузки машины:

$$T = \min \left\{ \max_j T_j \right\}.$$

Переменными задачи являются величины b_{ij}, причем

$$b_{ij}=\begin{cases}1, \text{ если алгоритм } i \text{ реализуется на машине } j;\\ 0, \text{ если алгоритм } i \text{ не реализуется на машине } j, \text{ сумма}\end{cases}$$

$$\sum_{j=1}^{n} b_{ij}=1 \text{ для всех } i=1, 2, \ldots, m.$$

При планировании необходимо найти двоичные значения величин b_{ij} при условии $b_{i_1 j} \neq b_{i_2 j}$ для $i_1 \neq i_2$, где $i_1, i_2 \in m$ и $j \in n$, и такие, что линейная форма достигает минимума.

Из постановки и условий задачи следует, что рассмотренный выше метод планирования и его модификации могут применяться для распределения по машинам только независимых алгоритмов, причем процессоры должны обладать различной производительностью и число задач должно строго соответствовать числу машин. Кроме того, методы планирования, основанные на целочисленном программировании, требуют для реализации на ЭВМ больших затрат времени и памяти, например при $n=m=5$ необходимо просматривать 1728000 различных планов.

Одна из модификаций рассмотренного выше метода предполагает предварительное упорядочение величин τ_{ij}, что позволяет упростить процедуру распределения независимых и равноправных в смысле приоритета задач по машинам.

Целевой функцией здесь выбрано минимальное значение общего времени решения задач, затрачиваемое при параллельной работе машин от начала ввода задач до окончания вывода результатов их решения.

Каждая из задач может быть выполнена на одной из машин различного или равного быстродействия.

Машины располагаются в списке по мере убывания их мощности. Матрица реализаций $|\tau_{ij}|$ строится так, что ее элементы удовлетворяют условию:

$$\tau_{i-1,\, j} \leqslant \tau_{i,\, j} \leqslant \tau_{i+1,\, j}.$$

При этом любая из задач выполняется на машине с номером K за время не большее, чем время ее реализации на машине $(K+1)$.

Решение поставленной задачи распределения проще получить при переходе к двойственной задаче линейного программирования, которая состоит в нахождении таких целых чисел $x_i (i=1, 2, \ldots)$ и $x_j (j=1, 2, \ldots)$, когда величина $\sum_{i=1}^{m} x_i + \sum_{j=1}^{m} x_j$ максимальна при $x_i + x_j \leqslant \tau_{ij}$.

Решение этой задачи может быть получено на одной из машин системы и не требует больших затрат машинного времени.

Значительный интерес представляет метод нахождения оптимального плана обработки задач, обеспечивающий глобальное строго экстремальное решение применительно к специализированной ВС. Сущность метода заключается в применении принципа опти-

мальности и интерпретации процесса планирования как процесса управления. Предполагается решение задачи распределения по машинам независимых квантов, т. е. квантов, граф которых обладает нулевой связностью.

Пусть имеется m независимых программ:

$$X = \{X_1, X_2, \ldots, X_j, \ldots, X_m\},$$

каждая из которых состоит из упорядоченной последовательности подпрограмм x_i:

$$X_j = \{x_1, x_2, \ldots, x_i, \ldots, x_n\}.$$

Положим, что ВС состоит из M машин ($k = 1, 2, \ldots, M$), каждая из которых в данный момент времени может обрабатывать только одну из подпрограмм j-й программы. Каждая подпрограмма любой программы в данный момент обрабатывается только одной машиной и каждая подпрограмма, начатая на данной машине, обрабатывается без перерыва и заканчивается на этой же машине.

Таким образом, символ x_{ijk} означает, что i-я подпрограмма j-й программы выполняется на k-й машине.

Обозначим состояние системы символов S. Под состоянием системы на данном этапе распределения η будем понимать возможный вариант распределения выполнения программ x_j по машинам на этом этапе. Таким образом,

$$S_\alpha^{(\eta)} = S_\alpha^{(\eta)}[k(j)]; \quad \eta = 0, 1, 2, \ldots, N;$$

$$\alpha = 1, 2, \ldots, r$$

означает, что k-я машина на η-м этапе выполняет j-ю программу в соответствии с вариантом распределения номера α.

Система на этапе η может перейти из одного состояния в другое при воздействии в конце этапа одного из возможных управлений: где U — заданное множество управлений;

$$U_\mu \in U, \quad \mu = 1, 2, \ldots, S,$$

μ — номер управляющего воздействия.

Переход системы из одного состояния в другое осуществляется по заданному закону:

$$S_\alpha^{(\eta+1)} = \varphi(U_\mu^{(\eta)}),$$

где $U_\mu^{(\eta)}$ — управление на η-м этапе.

Предположим теперь, что известно τ_{ijk} — среднее время, затрачиваемое k-й машиной на выполнение i-й подпрограммы j-й программы. Тогда задачу нахождения оптимального плана реализации m программ на ВС можно сформулировать следующим образом: требуется найти такую последовательность из $(N-1)$ управлений системы:

$$\vec{U} = \{U_\mu^{(0)}, U_\mu^{(1)}, \ldots, U_\mu^{(N-1)}\}$$

при обработке m программ, чтобы общее время выполнения всех программ на ВС было бы минимальным:

$$\tau(\vec{U}) = \min.$$

Метод определения оптимального плана реализуется на основе принципа оптимальности динамического программирования.

Одним из методов планирования загрузки, допускающим планирование зависимых задач, является эвристический алгоритм Шварца. Применительно к двум машинам (процессорам) предусматривается параллельное выполнение алгоритмов задач. Алгоритм задачи представляется ориентированным графом. Операторы исходного алгоритма могут быть как свободными, допускающими их выполнение на любой машине, так и связанными с одной из них. На основе исходного алгоритма строится матрица операторов, на которой, не прибегая к перебору вариантов разбиения алгоритма, выполняется упорядочение последовательности реализации операторов. Данный алгоритм позволяет получить на двух машинах такое время решения задачи, которое близко к минимально возможному за счет более плотного графика загрузки.

В заключение отметим, что ни один из рассмотренных выше методов не обладает универсальным свойством оптимальности при возможных комбинациях внешних условий. Задача упорядочения и распределения в настоящее время — быстро развивающаяся область исследования операций. Выбор путей решения этой задачи остается специфическим для каждой конкретной вычислительной системы.

ВОПРОСЫ К ГЛАВЕ

1. Перечислите преимущества и недостатки однопрограммного режима работы.
2. Назовите сущность, преимущества и недостатки мультипрограммирования.
3. Назовите сущность, преимущества и недостатки параллельной обработки информации.
4. Характеристика режима работы в разделенном времени.
5. Перечислите основные принципы классификации дисциплин обслуживания заявок.
6. Преимущества и недостатки бесприоритетных линейных дисциплин обслуживания заявок.
7. Преимущества и недостатки бесприоритетных циклических дисциплин обслуживания заявок.
8. Назначение, основные разновидности и характеристика дисциплин с фиксированными приоритетами.
9. Дайте определение основных характеристик функционирования модели в многопрограммном режиме работы.
10. Преимущества и недостатки дисциплин с динамическими приоритетами.

Глава 4

СТРУКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

4.1. Назначение программного обеспечения

Программным обеспечением вычислительной машины (или комплекса машин) называется совокупность программ регулярного применения, описаний и инструкций по их применению, предназначенная для технической эксплуатации ЭВМ и ее использования.

Необходимость разработки программного обеспечения диктуется рядом обстоятельств, среди которых наиболее существенные: упрощение общения оператора с машиной и машины с оператором; сокращение сроков прохождения задач от их постановки до машинной реализации; повышение эффективности использования ЭВМ и многомашинных комплексов.

По задачам и функциям, выполняемым различными элементами программного обеспечения, можно выделить две большие группы: общее и специальное программное обеспечение.

К *общему программному обеспечению* относится совокупность программ, описаний и инструкций, предназначенных для автоматизации трудоемких технологических этапов разработки алгоритмов и программ, а также для организации и контроля вычислительного процесса на машине во время ее функционирования.

Специальное программное обеспечение представляет собой совокупность программ решения конкретных задач из различных сфер применения.

Назначение программного обеспечения — согласование различных применений и способов работы вычислительной системы с целью увеличения ее производительности, повышения производительности труда программиста, адаптируемость программ к изменяющимся ресурсам, возможности расширения программного обеспечения.

Высокий уровень производительности вычислительной системы обеспечивается операционной системой двумя способами. Во-первых, в режиме пакетной обработки операционная система помогает оператору в выполнении подготовительных действий, которые совмещаются с исполнением ранее воспринятых заданий. Во-вторых, при мультипрограммном режиме операционная система распределяет и перераспределяет ресурсы машины между различными программами, сводя простои ресурсов к минимуму.

Повышение производительности труда программиста происходит за счет предоставления ему разнообразных средств программирования: включения большого числа языков программирования, библиотек наиболее употребительных программ, удобных средств ввода-вывода, отладки, а также средств удобного оформления работ.

Адаптируемость программ к изменяющимся ресурсам обеспечивается тем, что операционная система имеет средства обслуживать

большой диапазон машинных конфигураций. Кроме этого, операционная система позволяет строить программы, независимые от устройств ввода-вывода, что упрощает проблему переработки программ, связанных с заменой устройств ввода-вывода.

Программное обеспечение можно использовать на вычислительных системах с новыми внешними устройствами, для новых областей применения, с новыми программами по мере их появления. При предъявлении дополнительных требований в операционную систему могут быть добавлены средства, реализующие дополнительные возможности, добавлены компиляторы с новых языков.

4.2. Состав общего программного обеспечения

Общее программное обеспечение современных вычислительных машин включает в себя алгоритмические языки и языки программирования с соответствующими компилирующими системами; операционную систему, осуществляющую общую организацию процесса обработки информации и связь ЭВМ с пользователем; обслуживающую систему для отладки, контроля и диагностики неисправностей в ходе работы ЭВМ.

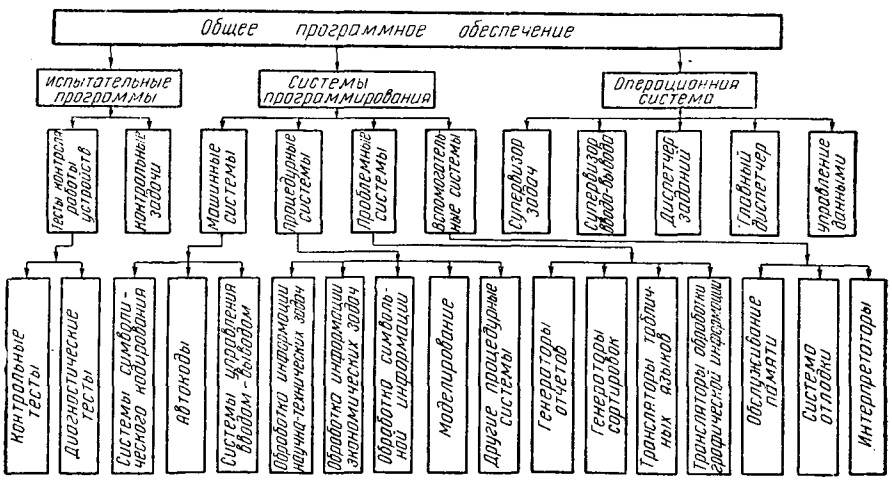


Рис. 4.1. Классификационная схема системы

Классификационная схема программного обеспечения приведена на рис. 4.1.

Общее программное обеспечение по характеру использования отдельных элементов разделяется на испытательные программы, системы программирования и операционную систему.

Испытательные программы предназначены для проверки работоспособности, наладки и технической эксплуатации вычислительных систем. Эти программы принято разделять на контрольные задачи

и тесты. Контрольные задачи предназначены для комплексной проверки работоспособности системы при приемо-сдаточных испытаниях, а также после проведения соответствующих профилактических работ (годовая и полугодовая профилактика). Тесты выполняются обычно по отдельным устройствам машины и предназначены для выявления их работоспособности. Оказалось выгодным разделение тестов на контрольные, устанавливающие факт наличия неисправности, и диагностические, локализующие отказавший элемент схемы.

Система программирования должна освободить программиста от необходимости изложения проблемы на машинном языке введением специального языка более высокого уровня. Система программирования должна включать в свой состав по меньшей мере два компонента:

входной язык системы программирования;

программу, обеспечивающую перевод с входного языка на машинный.

Система программирования позволяет снизить трудоемкость разработки программ для ЭВМ в той части, которая связана непосредственно с написанием программы. Операционная система предоставляет программисту совокупность приемов и процедур для работы на машине.

Под *операционной системой* понимается совокупность программ, управляющих ходом работы машины, идентифицирующих программы и данные и осуществляющих связь между машиной и оператором.

С функциональной точки зрения операционная система состоит из управляющих и обрабатывающих программ. Назначение управляющих программ — организовать обработку входных заданий, данных и задач. Можно выделить три основных функции управляющих программ: управление заданиями, управление задачами, управление данными.

К обрабатывающим программам относятся проблемные программы и те программы операционной системы, которые выполняются в состоянии процессора P1. Примерами обрабатывающих программ являются также компиляторы, отладочные средства, генераторы и некоторые пакеты прикладных программ.

Часть управляющей программы, которая организует прием заданий из входного потока заданий, осуществляет их контроль, подготовку к выполнению и запуск, называется УПРАВЛЕНИЕМ ЗАДАНИЯМИ.

Управление заданиями выполняет следующие функции: анализ входного потока заданий, запись входного потока на устройства с прямым доступом, подготовку задания к выполнению, распределение устройств ввода-вывода, управление установкой носителей (например, рулонов бумаги), подготовку к запуску программ с контрольной точки, общее планирование порядка выполнения работ.

Анализ входного потока заданий заключается в определении начала первого задания, контроля правильности управляющих операторов и их следования, наличия конца описания одного задания и начала следующего или конца заданий.

Запись входного и выходного потока на устройства с прямым доступом осуществляется для ускорения прохождения работ.

Подготовка задания к выполнению заключается в проверке всех имен отдельных частей программы, из которой состоит задание, построении ряда таблиц, необходимых для правильного выполнения программы, предоставлении ресурсов заданию.

Распределение устройств ввода-вывода производится с целью более эффективного их использования, а также для обеспечения независимости программ от конкретных физических адресов устройств с помощью метода логических устройств. Суть метода в том, что устанавливается стандартный набор символических имен устройств, которые программист использует в исходной программе. Таким образом, в программе указывается логическое устройство, а не конкретный физический адрес требуемого устройства. УПРАВЛЕНИЕ ЗАДАНИЯМИ содержит процедуры назначения логическим устройствам физических адресов, механизм поддержания установленных назначений во время обработки потока заданий и их использования при обращении к устройствам ввода-вывода.

Управление установкой носителей заключается в том, что оператору заранее выдается информация о том, какие необходимо включить устройства ввода-вывода, какие надо установить тома для бесперебойной работы системы. Томом называется стандартный блок внешней памяти, например катушка магнитной ленты, пакет дисков и т. д.

Подготовка к запуску программ с контрольной точки означает те же действия, что и подготовка программы к выполнению. Операционная система позволяет программисту указать в программе контрольные точки, начиная с которых можно возобновить выполнение пограммы при необходимости.

Общее планирование порядка выполнения работ определяет тип мультипрограммирования. Существуют три типа мультипрограммирования, которые определяют три основные конфигурации операционной системы: операционную систему с первичной управляющей программой (РСР); операционную систему, которая обеспечивает мультипрограммирование с фиксированным числом задач (*MFT*); операционную систему, которая обеспечивает мультипрограммирование с переменным числом задач (*MVT*).

Часть управляющей программы, которая контролирует выполнение задачи на всех этапах, начиная с момента ввода задания для выполнения и до получения результатов работы, называется УПРАВЛЕНИЕМ ЗАДАЧАМИ.

Управление задачами, которые могут выполняться индивидуально или одновременно, осуществляют программы СУПЕРВИЗОРа ЗАДАЧ. СУПЕРВИЗОР включается в работу по сигналам преры-

вания и обеспечивает обработку прерываний; обработку сбоев внешних устройств; планирование работы каналов; обслуживание таймера, выполнение процедур, связанных с окончанием задания; связь с оператором; одновременное выполнение нескольких программ; загрузку программы в основную память для выполнения; создание контрольных точек и возможность возобновления обработки с контрольной точки; защиту памяти.

Обработка прерываний заключается в том, что СУПЕРВИЗОР распознает тип прерывания и передает управление соответствующей программе, которая осуществляет обработку. Программист полностью освобожден от необходимости обрабатывать прерывания.

При возникновении сигналов прерывания по программным сбоям СУПЕРВИЗОР дает возможность выбрать один из путей: снятие задания с выполнения с распечаткой или без распечатки основной памяти того раздела, в котором оно выполнялось; переход к подпрограмме потребителя, которая сама должна принять решение в создавшейся ситуации. Если прерывание замаскировано (запрещено), то в случае его возникновения система игнорирует это прерывание. Прерывание по машинному сбою приводит вычислительную систему в состояние ожидания. Дальнейшая работа может начаться с процедуры первоначальной загрузки системы.

Планирование работы каналов заключается в создании очередей к устройствам и каналам на основе запросов на операции ввода-вывода, в запуске операций ввода-вывода и обработке прерываний ввода-вывода, возникающих при нормальном завершении операции, при обнаружении ошибки, достижении конца файла.

Обслуживание таймера означает слежение за временем суток, которое проблемная программа может запросить в любой момент своего выполнения, печать времени начала и окончания выполнения задания, управления интервалом времени, запрошенным программой.

По завершении выполнения задания (аварийного или нормального) управление передается СУПЕРВИЗОРу для реализации системных действий, связанных с окончанием задания. При этом в случае аварийного окончания может быть произведена распечатка основной памяти.

Управление задачами предоставляет оператору возможность управления работой системы, обеспечивая его двусторонней связью. С одной стороны, система дает оператору информацию, позволяющую следить за ходом вычислительного процесса, и требует отвечать на запросы в тех ситуациях, когда работа не может быть продолжена без вмешательства оператора. С другой стороны, система принимает ответы оператора на поставленный запрос, а также другие директивы оператора системы.

Мультипрограммирование обеспечивает выполнение как одиночной независимой программы, так и нескольких одновременно. Оперативная система, в которой в каждый момент времени выполняется одна задача, нуждается в относительно простом управлении. Каж-

дая задача получает нужные ей ресурсы, когда в них возникает необходимость.

Система, которая обрабатывает несколько задач одновременно, нуждается в способе идентификации задач, присвоении им приоритетов, распределении ресурсов между задачами в соответствии с их приоритетами.

Загрузка программ в основную память означает определение местоположения программ в одной из системных библиотек и вызов ее в основную память.

Контрольные точки создаются программистами ,чтобы иметь возможность временно прекратить выполнение программы и возобновить ее выполнение не с начала, а с данной контрольной точки.

Защита памяти предназначена для предохранения программ и соответствующих им данных от вмешательства других программ. Защита памяти используется при мультипрограммном режиме и обеспечивается проверкой ключей защиты памяти.

Часть управляющей программы, представляющая программисту стандартные процедуры для управления данными и обеспечение к ним доступа, называется УПРАВЛЕНИЕ ДАННЫМИ. УПРАВЛЕНИЕ ДАННЫМИ предназначено для идентификации, организации, хранения и обращения к информации, которая должна быть обработана. Функции управления данными выполняет система управления вводом-выводом.

На ранних этапах развития программного обеспечения системы управления вводом-выводом обеспечивали передачу данных и обработку меток. Обычно они состояли из программ, которые управляют буферами, интерфейсом между аппаратурой и обращением к снабженным метками магнитным лентам.

Некоторые системы включали средства чтения и записи в устройства с прямым доступом и действия, управляемые с удаленных терминалов. Имелись также средства для выбора и формирования программ из библиотеки программ, но, как правило, они не входили в системы управления вводом-выводом.

УПРАВЛЕНИЕ ДАННЫМИ операционной системы обеспечивают все эти функции.

УПРАВЛЕНИЕ ДАННЫМИ рассматривает не отдельные устройства, а классы устройств. Данные из устройств с прямым доступом, удаленных терминалов, магнитных лент, организованные последовательно или как в библиотеке программ, могут быть запрошены, по существу, одним и тем же способом.

УПРАВЛЕНИЕ ДАННЫМИ обеспечивает: распределение вспомогательной памяти на устройствах с прямым доступом, размещение данных, их защиту как от случайной записи вне указанных границ, так и от запрещенных доступов к ценным файлам при помощи паролей.

4.3. Специальное программное обеспечение

Специальное программное обеспечение состоит из комплектов программ, добавляемых к общему программному обеспечению. Отличительными особенностями компонент специального программного обеспечения является ориентация на сравнительно узкий круг решаемых задач и большое их разнообразие.

Пакеты прикладных программ функционируют под управлением операционной системы.

Пакеты прикладных программ являются мощным средством автоматизации программирования. В период создания автоматизированных систем управления, когда программирование приобретает промышленный характер, разработка пакетов прикладных программ приобретает важное значение. Они являются одним из мощных источников развития программного обеспечения вычислительных систем и служат для ускорения и облегчения внедрения вычислительной техники в различные сферы хозяйственной деятельности.

Не следует путать термины: пакетная обработка и пакеты программ. Пакетная обработка представляет собой режим работы системы, когда на вход системы подается группа заданий (пакет заданий), оформленных как единое целое. Пакет программ — комплекс программных средств, предназначенных для решения какого-либо класса задач.

4.4. Возможности программиста

Пользователь вычислительной системы обычно имеет дело с обрабатывающими программами, т. е. с системами программирования и пакетами прикладных программ.

Поэтому входным языком в систему для него служит тот язык, на котором он пишет свои программы, например КОБОЛ или ФОРТРАН. Все остальные действия по переработке программы на исходном языке выполняют отдельные компоненты программного обеспечения.

Пользователь может:

писать программы, которые не зависят от устройства ввода-вывода или характеристик режима работы. Запросы на устройства ввода-вывода могут быть точно определены без изменения программы, когда начато ее выполнение;

помещать информацию (программы или массивы данных) в библиотеку системы без определения или наблюдения за идентификацией используемого устройства вспомогательной памяти, и затем получать доступ к информации только установлением символического обращения к ней. Система автоматически планирует размещение информации, выдает команду оператору — ввести информацию непосредственно в машину — и обеспечивает программиста справочным материалом, который потом используется для поиска и обработки информации должным образом;

запоминать часто используемые последовательности управляющих операторов, которые определяются в дальнейшем как «каталогизированные процедуры», и легко вызывать их для использования;

получать результаты вычислений вскоре после передачи их для выполнения, поскольку безостановочная работа системы не требует, чтобы задания задерживались до тех пор, пока их не накопится некоторая группа;

составлять эффективные программы, которые равномерно используют ресурсы системы. Программы могут быть разделены на простые подпрограммы, которые выполняются одновременно под управлением операционной системы. Таким образом, нет необходимости составлять программы сложной конструкции, чтобы достичь тех же результатов;

делить проблемы на отдельные задачи и записывать каждую на языке, который для нее больше подходит;

делить большие программы на небольшие сегменты, которые во время выполнения будут вызываться на одно и то же место для того, чтобы сократить пространство основной памяти;

легко проверять и изменять программы и данные;

делать выбор между непосредственным выполнением скомпилированных программ (или частей программ) или запоминанием их на вспомогательных устройствах с тем, чтобы позже использовать их с другими результатами работы компилятора или ассемблера. Система может сохранять и перемещать такие программы, если только даны их имена, и может передавать им управление различными способами;

использовать одну и ту же программу без повторного компилирования ее, даже если во время генерации системы были добавлены некоторые дополнительные возможности к управляющей программе, то есть при изменении операционной системы скомпилированная программа может выполняться без дополнительной переработки.

4.5. Требования и принципы проектирования программного обеспечения

Программное обеспечение представляет собой совокупность сложных систем управления, ориентируемую на использование вычислительных систем для различных применений. Это вызывает ряд требований к проектированию отдельных компонентов программного обеспечения. Принятые соглашения по некоторым вопросам реализации отдельных компонентов программного обеспечения накладывают ограничения на построение алгоритмов и программ для различных применений.

Рассмотрим основные требования, предъявляемые к проектированию отдельных компонентов программого обеспечения. Для реализации возможностей, предоставляемых пользователю, необходимо, чтобы система имела модульную структуру.

Большую систему будет создавать большой коллектив специалистов. Разбиение большой системы на отдельные, поддающиеся

обозрению части, упрощает разработку отдельных частей, но требует четкой организации работ. Необходимо наладить четкий график разработки отдельных частей, установить и определить все связи между ними. Определив интерфейс между различными частями, можно будет проводить работу различными людьми независимо от состояния дел в смежной области.

Модульность системы дает и другие преимущества, наиболее важными из которых являются возможность развития системы, легкость ее поддержания и гибкость.

Требование возможности развития законченной операционной системы возникает по двум причинам. Во-первых, спроектированную систему можно использовать как базовую для построения более мощных систем. Затраты на развитие системы будут уменьшаться, и освобожденный персонал может быть использован на других работах. Это приведет к тому, что более производительные системы могут быть созданы с меньшими усилиями.

С другой стороны, в силу модульности системы появляется возможность использовать ее для конкретных нужд пользователя без изменения существующего задела программ программного обеспечения. Появляется возможность генерировать требуемую операционную систему, используя только необходимые возможности системы и наращивая по мере необходимости.

Требование легкости поддержания и обслуживания возникает из-за необходимости уменьшать количество обслуживающего персонала системы. Модульность системы позволяет стандартизировать обслуживание отдельных компонентов и иметь минимальное количество специалистов по отдельным частям системы.

Система должна быть гибкой. Ее могут использовать потребители в различных сферах применения. Это требование также удовлетворяется модульностью системы, возможностью расширения, выбором нужных функций из имеющихся в системе, а также за счет независимости программы пользователя от имеющегося периферийного оборудования.

Следующее требование, предъявляемое к системе, — совместимость с другими системами. Совместимость должна обеспечить обмен программами и данными между различными моделями одной системы и машинами этого же класса других систем. Совместимость достигается за счет наличия стандартной системы команд, путем введения понятия логического и физического уровней ввода-вывода данных, а также за счет предоставления пользователю стандартных возможностей по эксплуатации системы.

Значительное внимание должно быть уделено выявлению ошибок. С одной стороны, необходимо выявить ошибки пользователя, возникшие при составлении программы и при оформлении ее для ввода и выполнения в машине. С другой стороны, необходимо иметь средства проверки надежности работы отдельных компонентов операционной системы и машины.

В основу проектирования программного обеспечения были поло-

жены три основных принципа: параметрическая универсальность, функциональная избыточность и функциональная избирательность.

Принцип параметрической универсальности использовался программистами, когда надо было учесть в программе изменяющееся число компонентов. Так как различные конфигурации машины имеют неодинаковое число устройств ввода-вывода, различный объем основной памяти и других ресурсов, учесть эти изменения возможно, написав программы с параметрами. Этот принцип, положенный в основу большинства программ отдельных компонентов программного обеспечения, позволяет различным потребителям использовать необходимые компоненты для своих применений.

Функциональная избыточность. В практике программирования часто встречается случай, когда, получив выигрыш во времени работы программы, мы теряем в объеме памяти, а выиграв в объеме памяти, теряем время. В этом случае, когда нужно оптимизировать производительность при наличии двух и более противоречивых целей, создаются две или более программы, реализующие одну и ту же функцию, но работающие по разным принципам. Этим методом часто пользуются при создании трансляторов. Он нашел также широкое применение при проектировании операционных систем.

Функциональная избирательность. Ядро операционной системы включает минимально необходимый набор возможностей. Дополнительные функциональные возможности, рассчитанные на различные сферы применения и конфигурации машин, оформляются как необязательные и включаются в конкретную операционную систему по требованию пользователя во время генерации системы. Вызвано это различными причинами, важнейшая из которых — каждая дополнительная возможность увеличивает объем основной памяти, занимаемой операционной системой. Наиболее важными из дополнительных возможностей управляющих программ являются возможности, относящиеся к различным режимам планирования заданий и управления задачами; разные методы распределения основной памяти, защита основной памяти, выполнение заданий в соответствии с их приоритетами. Этот принцип нашел отражение и в том, что с каждого входного языка, входящего в состав программного обеспечения, имеется несколько компилирующих систем.

ВОПРОСЫ К ГЛАВЕ

1. Что понимается под программным обеспечением?
2. Определите состав и назначение отдельных компонентов программного обеспечения.
3. Чем обеспечивается высокий уровень производительности системы и программистов?
4. Что понимается под системой программирования?
5. Назначение и задачи, решаемые операционной системой.
6. Назначение и методы использования испытательных программ.
7. Назначение и характеристика пакетов прикладных программ.
8. Назначение и основные функции управления заданиями.
9. Назначение и основные функции управления задачами.
10. Назначение и основные функции управления данными.

Глава 5

ДОКУМЕНТИРОВАНИЕ ЭЛЕМЕНТОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

5.1. Назначение и требования к технической документации

К технической документации на программное обеспечение относятся графические и текстовые документы, которые в совокупности определяют состав и функциональную структуру компонентов программного обеспечения и содержат необходимые данные для их разработки, отладки, контроля, приемки, эксплуатации и изучения.

Уже из самого определения вытекает многоцелевое назначение технической документации. Документация является одним из важнейших элементов в составе программного обеспечения, а ее выпуск представляет собой весьма трудоемкий процесс. Поэтому необходима система, обеспечивающая автоматический выпуск документации. Частично такие системы уже разработаны. Обеспечивается автоматический выпуск документации, необходимой для эксплуатации программ и изучения их обслуживающим персоналом. Как правило, все системы программирования обеспечивают выпуск листингов и текста программ на подходящих носителях для непосредственного размножения и использования.

Предпринимаются успешные попытки разработки программ, обеспечивающих составление и выпуск блок-схем. Делается это путем включения соответствующих макрокоманд в тело программы аналогично использованию отладочных средств. Все эти работы имеют большое значение в связи со значительным объемом, разнообразием видов и необходимостью внесения изменений в документацию.

Сложность решения этой задачи определяется необходимостью учета всех изменений и проведения связанных корректировок. При этом выпуск корректировочной документации (извещений на изменения) должен сопровождаться проверкой правильности проведения корректировок. Состав и виды документов зависят от технологии разработки и реализации компонентов программного обеспечения, так как документация сопровождает все этапы проектирования, эксплуатации и обучения. В связи с этим документация должна удовлетворять следующим основным требованиям.

Единство терминологии системы программного обеспечения. Терминологическое многообразие может привести к двусмысленности, непониманию и искажению смысла отдельных документов, затруднит понимание взаимосвязи между различными документами, усложнит их использование, а при внесении изменений может привести к значительным искажениям, исправление которых может потребовать привлечения высококвалифицированных специалистов.

Единство номенклатуры и наименования документов. Необходимость данного требования вытекает из тех условий, что и необходи-

мость единства терминологии, и усугубляется еще тем, что отдельные компоненты программного обеспечения могут разрабатываться разными организациями. Это требование вытекает из нужд координирования разработок.

Единая система обозначений в документах. Данное требование означает, что однотипные документы на различные компоненты программного обеспечения должны быть тождественными независимо от того, в одной организации они создаются или в разных. Это требование возникает также из нужд однозначного понимания документов на всех стадиях разработки, эксплуатации и обучения.

Идентичность документов независимо от места их разработки. Это требование объясняет все вышеизложенные. Разработка компонентов программного обеспечения производится совместно многими организациями. В блок-схеме, например, независимо от организации, в которой она выполнялась, независимо от компонента, на который она составлялась (элемент операционной системы, пакета программ, компилирующей системы), терминология, названия документа, обозначения в документе должны быть идентичными.

Единые правила учета и хранения документации. Это требование накладывает жесткие ограничения на порядок передачи извещений на изменения, строгое внесение их во все документы, правильную эксплуатацию системы программного обеспечения. Это одно из самых важных требований, так как внесение изменений в одни документы и невнесение в другие может привести к разрушению отдельных компонентов программного обеспечения.

5.2. Виды документов

Исходя из опыта практического применения технической документации можно определить следующие виды основных документов программного обеспечения.

Блок-схема представляет собой документ, содержащий подробную диаграмму программы, потока данных, этапов обработки и другие данные. Блок-схемы предназначены для представления логической схемы программы, логической схемы системы, представления взаимосвязей потоков данных в различных компонентах системы.

При составлении логической схемы системы внимание обращается в основном на обрабатываемые физические документы (учитывая носители) и на элементы, используемые при сбработке, и на подробности выполнения операций. На рис. 5.1а приведены три основных символа, используемых для начертания любых логических схем системы. Прямоугольник соответствует любому виду обработки данных, трапеция — любой операции ввода-вывода, стрелки определяют порядок выполнения действий. На рис. 5.1б приведены дополнительные символы логических схем систем, раскрывающие некоторые действия. Дополнительные символы являются необязательными.

При составлении блок-схемы программы (или логической схемы программы) особое внимание обращается на подробное формальное описание процесса обработки данных с учетом специфики реализации на вычислительной машине. Поэтому для изображения блок-схемы используются несколько иные символы, чем на логических схемах системы. Эти символы представлены на рис. 5.2. В блок-схеме программы можно воспользоваться основными символами схем системы (рис. 5.1) и специфичными символами схем программы (рис. 5.2). Блок-схемы программы могут составляться с разной степенью детализации.

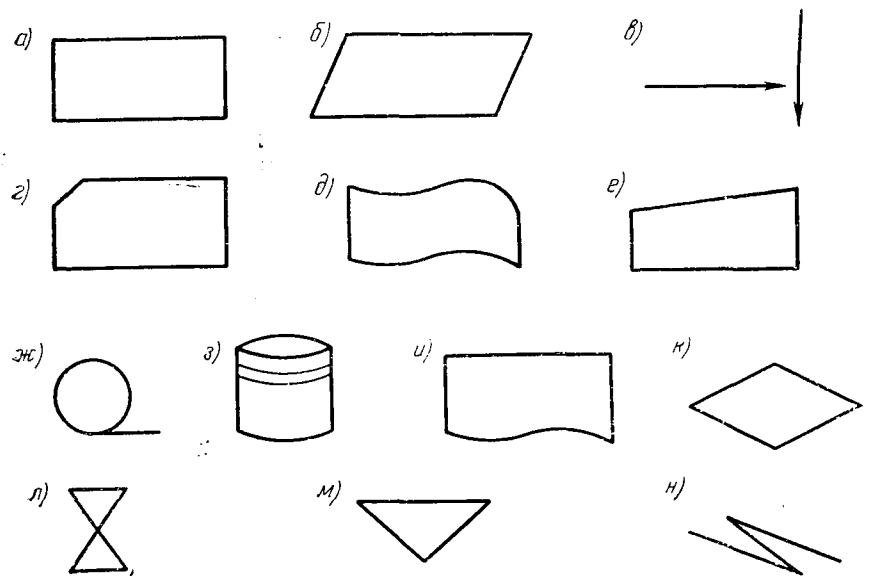


Рис. 5.1. Символы логических схем системы:

основные символы логических схем системы: *а* — обработка, *б* — ввод-вывод, *в* — последовательность выполнения действий; дополнительные символы логических схем: *г* — перфокарты, *д* — бумажная лента, *е* — пульт, *ж* — магнитная лента, *з* — устройство с произвольной выборкой, *и* — отпечатанные документы, *к* — разветвление, *л* — сортировка-объединение, *м* — слияние, *н* — линии связи

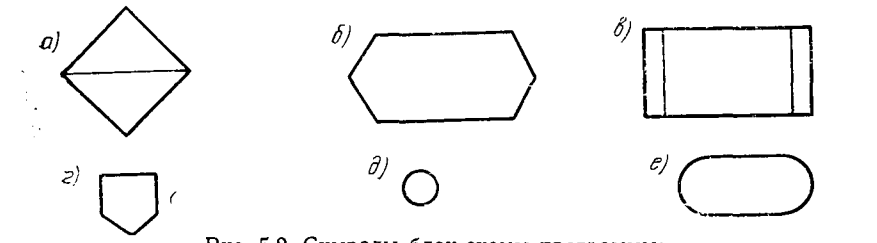


Рис. 5.2. Символы блок-схемы программы:

а — сортировка; *б* — модификация программы; *в* — подпрограмма; *г* — межстраничный соединитель; *д* — соединитель; *е* — точка начала, конца или прерывания

Листинг представляет собой документ, являющийся распечаткой компилятора в процессе трансляции программы, написанной на входном языке. Вид листинга применительно к ССК М-32 «Минск-32» приведен на рис. 5.3.

```
15.3.72                                                    ССК Минск-32

          Идентификатор прим 1
**        —70 00 0 0000 0 1000
         область      1                001 010          БАЗ   0
**        — 70 00 0 0030 0 1000
0 000000—0 000002                     001 020  БЛ1     РЗВ   3
                                      001 050  БЛ4     РИП   16
                                      001 060        * СУ    4 ; +16
0 000003 —17 00 0 0404 0 0024         *              СУ    4 ; Л, 000
0 000004 —20 01 1 0400 0 0023         001 065          ПАИ   : ИНД1 ; Н
0 000005 —10 00 2 0001 1 0000         001 070          П     МАС ; РАБ
0 000006   21 01 2 0002 1 0000       001 080  БЛ3     ВФ    : ИНД1 ; МАС + 1 ; РАБ
0 000007 —32 00 0 0010 0 0011         001 090          ИМН   БЛ2
0 000010 —10 01 2 0002 1 0000         001 100          П     : ИНД1 ; МАС + 1 ; РАБ
0 000011 —20 01 0 0400 0 0021         001 110  БЛ2     СИ    : ИНД1 ; АДР
0 000012 —22 00 0 0006 2 0000         001 120          ИС    БЛ3 ; ПАР
0 000013 —10 00 1 0000 0 0022         001 121          П     РАБ ; Р
0 000014 —60 00 0 1000 1 2401         001 122          ЗАКР  ПЧ , 1
                                      001 123        * ЛСДЗ  +2 ; Р
```

Рис. 5.3. Вариант листинга

Листинг содержит все необходимые пояснительные показатели (идентификатор, дата трансляции, номер листа), исходную и транслированную программы. При этом исходная программа для удобства работы с листингом печатается строго против соответствующих точек в рабочей программе в разделе пояснений. Этот документ может быть очень полезен при отладке программы и поиске ошибки.

Процесс трансляции (см. главу 9) сопровождается выводом ряда специфических документов. Применительно к транслятору МЭСИ-2 (рис. 5.4) рассмотрим распечатку исходной программы, записанной на алгоритмическом языке АЛГЭК-С, и таблицу контроля синтаксиса исходной АЛГЭК-программы (ИАП).

ИАП записывается на стандартных бланках, включающих десять строк (номера 00, 10, 20, ..., 80, 90). Четырехзначный десятичный номер строки в распечатке содержит также номер листа (две старшие цифры). Наличие строк в распечатке позволяет легко решить задачу целеуказания места синтаксической или семантической ошибки от начала строки, содержащей ошибку. Распечатка программы производится с учетом ее структуры, т. е. символ **КОНЕЦ** всегда печатается под «своим» символом **НАЧАЛО.** Все основные символы — слова языка — подчеркиваются при выдаче на печать.

Таблица контроля синтаксиса ИАП печатается после завершения проверки всей исходной программы в случае ошибок. Одной ошибке в ИАП соответствует одна строка таблицы контроля синтаксиса. В целях облегчения поиска по ошибочной строке транслятор сооб-

```
0100    ААМ:
        НАЧАЛО ЦЕЛЫЙ К, Р; ВЕЩЕСТВЕН М2, Н2, Л1; ВЕЩЕСТВЕН МАССИВ Х1 [10], У1 [10]; ПРОЦЕДУРА СУММА (Х, У. М, Н, И, Ж, Л);
0110        ЗНАЧЕНИЕ М, Н; ВЕЩЕСТВЕН МАССИВ Х. У; ЦЕЛЫЙ М, Н, И, Ж; ВЕЩЕСТВЕН Л;
        НАЧАЛО ВЕЩЕСТВЕН М1, Н1;
0120        М1 : = 1;  Л : = 0;  ДЛЯ  И : = 1 : Н + 1 ЦИКЛ М1 : = М1 × (Л — У [И]); ДЛЯ И : = 1 : Н + 1
0130        ЦИКЛ
          НАЧАЛО М1 : = Х [И] × М1; Н1 : = Л — У [И]; ДЛЯ Ж : = 1 : Н + 1 ЦИКЛ
0140            ЕСЛИ И ≠ Ж ТО НА М; Л : = Л + М1/Н1; М : Н1 : = Н1 + (У [И] — У [Ж]);
          КОНЕЦ
        КОНЕЦ;
0150    БИБЛИОТ ('ВВОД1', '1', Н); БИБЛИОТ ('ВВОД3', '2', Х1); БИБЛИОТ ('ВВОД3', '2', У1);
0160    СУММА (Х1, У1, М2, Н2, К, Р, Л1); БИБЛИОТ ('ВЫВОД1', '1', Л1)
    КОНЕЦ;
```

КОНТРОЛЬ СИНТАКСИСА ИАП

лист строка	предыд. символ	очеред. символ	пор. номер	код ошибки			
—0100	У	.	057	042	ОШИБКА В СПИСКЕ ФОРМАЛЬНЫХ ПАРАМЕТРОВ КОНТРОЛЬ СИНТАКСИСА ПРОДОЛЖАЕТСЯ С СИМВОЛА	,	ЛИСТ 0100
—0110	Х	.	011	047	ОШИБКА В СОВОКУПНОСТИ СПЕЦИФИКАЦИИ КОНТРОЛЬ СИНТАКСИСА ПРОДОЛЖАЕТСЯ С СИМВОЛА	;	ЛИСТ 0110

Рис. 5.4. Вариант распечатки ошибок

щает предыдущий символ, после которого был установлен факт наличия синтаксической ошибки, очередной символ, исследование которого натолкнуло на факт наличия ошибки, его порядковый номер от начала строки и код ошибки, по которому в руководстве можно найти полное описание ошибки и рекомендуемые приемы устранения.

Для оперативной работы по отладке программы краткое содержание ошибки печатается в разделе пояснений в таблице контроля синтаксиса. Второй строчкой по каждой синтаксической ошибке печатается сообщение о точке ИАП, с которой продолжен контроль после обнаружения данной ошибки. Важность такого сообщения диктуется, с одной стороны, необходимостью локализации непроверенного участка ИАП, с другой — попыткой за один выход на трансляцию найти как можно больше ошибок, допущенных в исходной программе. Указание на точку продолжения контроля дается координатой символа, с которого контроль продолжается, и номером строки, содержащей этот символ.

В условном примере рис. 5.4 допущены две ошибки, на которые синтаксический контроль отпечатал две строки с соответствующими комментариями.

После завершения контроля ИАП осуществляется проверка на предмет обнаружения ошибок. Если ошибки синтаксиса были обнаружены, то оператору сообщается об исключении из обработки данной задачи. Если ошибки синтаксиса отсутствуют, то процесс трансляции продолжается. Учебные материалы по системе программного обеспечения содержат информацию, необходимую для обучения всех категорий пользователей машины и программного обеспечения. К учебным материалам относятся различного рода руководства по пользованию, эксплуатации и изучению системы.

Кроме названных документов необходимы также документы, содержащие технические данные, подлежащие проверке при испытании и внедрении программного обеспечения. К ним относятся тесты проверки различных компонентов программного обеспечения и машины. При этом должны быть четко сформулированы все этапы проверки и конечные результаты каждого этапа. Необходимо также иметь документ, содержащий расчеты некоторых основных параметров. Например, расчет оценки памяти, расчеты временных параметров.

В зависимости от способа выполнения документов и характера их использования они подразделяются на оригиналы (подлинники) и копии. Оригиналы представляют документы, выполненные на любом материале и предназначенные для изготовления по ним копий. Они должны быть оформлены соответствующими подписями лиц, ответственных за выпуск документа. Копия представляет собой документ, выполненный способом, обеспечивающим его идентичность с подлинником, предназначенный для непосредственного использования при разработке, реализации, отладке и эксплуатации программного обеспечения.

5.3. Состав документации

Существует несколько способов классификации состава документации. В зависимости от комплектности документации на компонент программного обеспечения устанавливаются следующие категории: основные документы, основные комплекты документов, полный комплект документов.

Основной документ самостоятельно или в совокупности с другими записанными в нем документами полностью и однозначно определяет данный компонент программного обеспечения и его состав. К основным документам относятся блок-схемы и спецификации.

Основной комплект документов объединяет документы, относящиеся к данному компоненту программного обеспечения, но не содержит документы составных частей. К основному комплекту документов относятся макро-блок-схемы, описание логики программ и эксплуатационные документы.

Полный комплект документов включает в себя основные комплекты документов на компонент и основные комплекты документов на составные части данного компонента.

В зависимости от назначения и использования документы группируются внутри следующих библиотек: в справочной библиотеке системы, библиотеке описаний логики программ, библиотеке описаний прикладных программ, библиотеке документов обслуживания, библиотеке учебных материалов.

Справочная библиотека системы содержит общую информацию (библиографию, описание связей ввода-вывода, информационный бюллетень и др.), описание моделей вычислительной системы, описание устройств ввода-вывода, библиотеку описаний системных программ (общие руководства, каталоги, концепции управления и построения, описания отдельных компонентов программного обеспечения и др.), общие эксплуатационные материалы, язык управления заданиями, СУПЕРВИЗОР и управление данными.

Описание логики программы представляет документ, содержащий общую недетализированную блок-схему, описание функций отдельных модулей программы, их взаимосвязей внутри программы, а также взаимосвязи данной программы с другими программами. Описание логики представляет собой текстовый документ, позволяющий разобраться в общих положениях описываемого компонента, понять его назначение, функции и общую структуру построения.

Спецификация представляет документ, определяющий состав отдельной программы или компонента программного обеспечения. Компонент программного обеспечения может состоять из одной или более составных частей. Перечень этих составных частей с указанием входимости и взаимоотношений и определяет состав спецификации.

Ведомость спецификаций представляет документ, содержащий перечень всех спецификаций составных частей отдельного компонента или всего программного обеспечения с указанием их входимости.

Эксплуатационные документы предназначены для использования во время эксплуатации системы, обслуживания и отладки. К эксплуатационным документам относится документация оператора и документация пользователя. Документация оператора содержит описание процедур подготовки ввода-вывода, общее описание и последовательность операций при каждом прогоне программы, инструкцию для оператора, перечень всех нормальных и анормальных остановов с указанием условий останова и действий, которые должны быть предприняты. Документация пользователя содержит общее описание системы, с которой он работает, описание языков программирования, сервисных программ, генераторов сортировок, руководство для оператора, руководство для системного программиста и др.

Библиотека описаний логики программ содержит описания логики программ операционной системы, сервисных программ, компиляторов.

Библиотека описаний прикладных программ содержит описания пакетов прикладных программ.

Библиотека учебных материалов содержит описания курсов (план с указанием часов на каждую тему, списки обязательной и дополнительной литературы), самоучители по отдельным темам программного обеспечения, предназначенные для самостоятельного изучения, учебники по всем разделам программного обеспечения (архитектура вычислительной системы, программирование на ассемблере, программирование на исходных языках, входящих в программное обеспечение системы).

Библиотека документов обслуживания содержит различные справочники по отдельным документам, бланки кодирования и описания форм бланков для различных применений.

Для однозначного составления всех типов документов применяется специальная структура обозначений номеров форм документов, учитывающая тип библиотеки, разделы в данной библиотеке, регистрационный номер документа и порядковый номер редакции. Структура обозначений номера программы определяет вычислительную систему (ИБМ, ЕС), тип компонента программного обеспечения, тип применения (системный компонент, пакет программ, обслуживающая система и т. д.) и регистрационный номер программы.

ВОПРОСЫ К ГЛАВЕ

1. Назначение документации.
2. Основные требования к документации.
3. Назначение блок-схемы.
4. Назначение и состав листинга.
5. Описание логики программы.
6. Ведомость спецификаций.
7. Отличие полного комплекта документов от основного комплекта.
8. Назначение и состав справочной библиотеки.
9. Назначение, состав библиотеки описаний логики программы.
10. Библиотека описаний прикладных программ.

ЛИТЕРАТУРА

Бертэн Ж., Риту М., Ружие Ж. Работа ЭВМ с разделением времени. М., «Наука», 1970.

Вычислительная система IBM/360. Принципы работы. Перевод с англ. под ред. Штаркмана. М., «Советское радио», 1969.

Голубев-Новожилов Ю. С. Многомашинные комплексы вычислительных средств. М., «Советское радио», 1967.

Кокс Д. Р., Смит У. Л. Теория очередей. Перевод с англ. под ред. А. Д. Соловьева. М., «Мир», 1966.

Кофман А., Крюон Р. Массовое обслуживание, теория и приложения. М., «Мир», 1965.

Липаев В. В., Колин К. К., Серебровский Л. А. Математическое обеспечение управляющих ЦВМ. М., «Советское радио», 1972.

Мачулин В. В., Пятибратов А. П. Эффективность систем обработки информации. М., «Советское радио», 1972.

Мультипроцессорные вычислительные системы. Под ред. Я. А. Хетагурова, М., «Энергия», 1971.

Пашкеев С. Д. Основы мультипрограммирования для специализированных вычислительных систем. М., «Советское радио», 1972.

Поспелов Д. А. Введение в теорию вычислительных систем. М., «Советское радио», 1972.

Современное программирование. Мультипрограммирование и разделение времени. Сборник статей. М., «Мир», 1970.

Теория надежности и массовое обслуживание. Под ред. Б. В. Гнеденко. М., «Наука», 1969.

Уилкс М. Системы с разделением времени. М., «Мир», 1972.

Функциональная структура OS/360. М., «Советское радио», 1971.

Шерр А. Анализ вычислительных систем с разделением времени. М., «Мир», 1970.

РАЗДЕЛ 2

ИСПЫТАТЕЛЬНЫЕ ПРОГРАММЫ

Глава 6

ПРОГРАММНЫЙ КОНТРОЛЬ И ДИАГНОСТИКА НЕИСПРАВНОСТЕЙ

6.1. Назначение испытательных программ

Для успешной эксплуатации современных вычислительных машин необходимо располагать достаточно эффективными средствами контроля, способными обнаружить неисправность и локализовать место ее нахождения.

В настоящее время одной из основных проблем вычислительной техники является проблема надежности вычислительных машин, в которой можно выделить проблему повышения степени безотказной работы элементов, узлов и устройств машины; и проблему эффективности профилактического контроля и локализации неисправностей.

Первой проблеме уделялось и уделяется сейчас значительное внимание как в направлении повышения степени надежности отдельных элементов машинных схем за счет дальнейшего совершенствования технологии изготовления, так и за счет дублирования функции устройств в мультипроцессорных вычислительных системах.

В качестве испытательных программ используются в ряде случаев программы обычных вычислительных задач с известными результатами решения. Получение правильных ответов считается гарантией удовлетворительного состояния машины, получение неверных — свидетельством наличия неисправности. Недостаточность таких приемов контроля очевидна, так как многие программы (в том числе и испытательные) могут выполняться правильно при наличии неисправностей в машине. Кроме того, поиск неисправного элемента (детали) в современных ЭВМ оказывается довольно трудоемким делом.

Для машин, лишенных встроенного (аппаратного) контроля, испытательные программы (тесты) являются основным средством, с помощью которого производится их наладка, организуется проведение профилактических работ и поиск неисправностей в процессе эксплуатации. Для машин, оснащенных аппаратурой встроенно-

го контроля, тесты также необходимы, так как они обеспечивают проверку правильности сборки машин, создания тяжелых режимов работы устройств, в которых эти устройства проверяются как при наладке, так и периодически в процессе эксплуатации.

ЭВМ можно рассматривать как объект, состоящий из множества простых запоминающих и комбинационных элементов. Условимся считать каждый простой запоминающий элемент одноразрядной двоичной запоминающей ячейкой. Поведение этого элемента характеризуется выходным сигналом T' и его внутренним состоянием T. Сигналами и состоянием являются определенные значения электрических величин: напряжение, полярность импульсов, намагниченность и т. д., которым поставлены в соответствие единицы и нули.

Функционирование простого запоминающего элемента можно описать следующими зависимостями:

$$T' = \varphi'(T, x_1, x_2, \ldots, x_m);$$
$$T = \varphi(T, x_1, x_2, \ldots, x_m), \tag{6.1}$$

где $x_1, x_2, \ldots, x_m$ — множество входных сигналов. Как видно из зависимости (6.1), значение выходного сигнала и внутреннего состояния простого запоминающего элемента определяется не только множеством входных сигналов, но и тем исходным внутренним состоянием, которое зафиксировано до прихода входных сигналов.

Простой комбинационный элемент может быть описан одним из следующих нормальных уравнений:

$$z = x; \tag{6.2}$$

$$z = \bar{x}; \tag{6.3}$$

$$z = x \wedge y; \tag{6.4}$$

$$z = x \vee y, \tag{6.5}$$

где z — выходной сигнал, x, y — входные сигналы.

Комбинационные схемы, применяемые в машинах, реализуют обычно более сложные функции. Однако их функционирование может быть описано суперпозицией простых функций, реализуемых зависимостями (6.2)—(6.5). Например, вместо схемы, реализующей функцию

$$z = x_1 \wedge x_2 \wedge x_3 \wedge \ldots \wedge x_n,$$

можно рассматривать n—1 простых элементов:

$$z_1 = x_1 \wedge x_2;\ z_2 = z_1 \wedge x_3, \ldots;$$
$$z = z_{n-1} \wedge x_n.$$

Каждому из элементов соответствует какой-либо электронный прибор или его часть, включающая некоторое количество деталей и их соединителей. Элементы электронных схем теряют со временем

свои первоначальные характеристики. Это приводит к тому, что значения выходных сигналов элементов или их форма отличаются от номинальных, и в схеме появляется неисправность.

Схемой машины (устройства, узла) называется множество запоминающих и комбинационных элементов, связанных между собой в соответствии с реализуемыми алгоритмами. Каждая схема характеризуется своими входами (входными сигналами) и выходами (выходными сигналами).

Комбинацией входных сигналов для схемы называется один из возможных наборов сигналов-аргументов, включающий сигналы, выдаваемые запоминающими элементами схемы в фиксированный момент времени; сигналы, поступающие в тот же момент времени на входы схемы, соединенные с внешними источниками.

Если для некоторого набора входных сигналов x_0, y_0 элемента, реализующего функцию $z=f(x, y)$, вырабатывается выходной сигнал z_0 такой, что $z_0 \neq f(x_0, y_0)$, то имеют место ошибки в работе схемы. Максимально возможное количество ошибок каждого элемента определяется числом возможных различных наборов входных сигналов. О возникновении ошибки в запоминающем элементе можно говорить, если его внутреннее состояние изменяется без подачи на него входных сигналов.

Физической неисправностью в схеме называется любая электрическая или механическая неисправность узла, входящего в схему, появляющаяся в виде хотя бы одной ошибки одного из элементов схемы при одной из возможных комбинаций входных сигналов. Физическая неисправность некоторого элемента может проявиться либо в виде ошибки этого же элемента, либо в виде ошибки другого элемента, связанного с рассматриваемым.

Испытательная программа позволяет вместе с соответствующими входными данными обнаружить с определенной вероятностью элемент с физической неисправностью или группу элементов, в которую входит неисправный элемент. Испытательные программы обычно строятся применительно к конкретным устройствам.

Под режимом работы схемы понимаются условия работы, в которых оказывается схема при воздействии некоторого набора исходных данных. В соответствии с этим каждая из комбинаций входных сигналов определяет режим работы схемы в фиксированный отрезок времени.

Существуют два класса тестов: тесты, обнаруживающие неисправность, и тесты, диагностирующие неисправность. Тест, обнаруживающий неисправность (*контрольный*), имеет два возможных исхода:

1) схема (устройство) не содержит ни одной неисправности заданного класса;

2) схема содержит одну или несколько неисправностей заданного класса.

Тест, обнаруживающий неисправность, свидетельствует об исправном или неисправном состоянии схемы, но не сообщает инфор-

мации о месте появления неисправности. Тест, дающий эту информацию, называется *диагностическим.*

6.2. Конструирование испытательных программ на базе диагностических таблиц

Для правильной работы устройства достаточно, чтобы все его элементы находились в исправном состоянии. Поэтому решение задачи программного контроля целесообразно начать с выяснения возможных неисправностей конкретного устройства. Необходимо составить список наиболее вероятных неисправностей. При этом предполагается, что в контролируемом объекте может быть не более одной из перечисленных неисправностей. Если по какой-либо причине необходимо учесть наличие нескольких неисправностей, то в перечень нужно внести новую неисправность, которая включает одновременное существование этих нескольких неисправностей.

Все неисправности можно разделить на два класса: постоянные неисправности, которые преобразуют данную схему в новую, описываемую другими логическими формулами, и неисправности, появляющиеся от случайных причин. Контроль постоянных неисправностей организуется проще, а проверка случайных неисправностей во многом будет сводиться к контролю постоянных неисправностей.

Предполагаем, что исходная схема устройства и неисправности в большинстве случаев снова приводят к устойчивой схеме. Почти всякая программа, предназначенная для контроля какой-либо неисправности, реагирует и на многие другие неисправности. Для запоминания всех неисправностей, проверяемых отдельными программами, составляется диагностическая таблица (матрица неисправностей). В верхней строке матрицы выписываются названия неисправностей из перечня и для каждой из них отводится один столбец матрицы. Затем берется программа входных сигналов, которая при наличии первой неисправности и при отсутствии остальных выполняется неверно. В первом столбце строки ставится единица. После этого выдвигается гипотеза о наличии второй неисправности (и отсутствии других) и проверяется реакция этой программы на вторую неисправность. Если программа выполняется верно, то во втором столбце первой строки ставится нуль, если невѐрно — единица. Точно так же заполняются все клетки первой строки матрицы (табл. 6.1).

В результате в первой строке матрицы окажутся отмеченными единицами все те неисправности, из-за которых первая программа выполняется неверно, и нулями — те, которые не влияют на работу программы. Следовательно, если программа выполнялась верно, то неисправностей, отмеченных единицами, нет, но могут быть те, которым соответствуют нули первой строки матрицы.

После заполнения первой строки матрицы отыскивается столбец с минимальным номером, в котором еще нет единицы, и составляется программа, реагирующая на соответствующую неисправ-

ность. Затем в клетки второй строки матрицы проставляются нули и единицы в соответствии с методикой, принятой для первой строки. Эта процедура продолжается до тех пор, пока в каждом столбце не появится хотя бы одна единица. Если же некоторые столбцы остаются нулевыми при любом наборе программ, это означает неконтролируемость соответствующих неисправностей. Матрицу неисправностей в этом случае можно считать оконченной, ибо эти неисправности не отражаются на правильности работы машины.

Совокупность написанных программ образует тест, обеспечивающий проверку данного устройства. Если все программы выполняются правильно, то в устройстве нет неисправностей, отмеченных в списке.

В связи с тем, что в матрице отмечаются неисправности, влияющие на каждую отдельную программу, вне зависимости от того, как они действовали на другие программы, каждая из программ теста не должна использовать результаты других программ. В этом состоит основное требование к программам теста. Вторым требованием к программам тестов является ограничение их длины, т. е. минимальное количество входных сигналов, так как для длинных программ труднее заполнять матрицу неисправности.

Последнее требование — каждая программа должна информировать (выдавать на печать) о результатах проверки (правильное или неправильное прохождение теста).

Алгоритм построения минимального контрольного теста

Матрица неисправностей может быть использована для минимизации количества программ испытаний. Действительно, если исключить из матрицы столбцы, соответствующие неконтролируемым неисправностям, то можно считать, что логическая сумма всех строк, рассматриваемых как двоичные числа, равна числу, сплошь состоящему из единиц. Из матрицы можно удалить несколько строк так, что логическая сумма оставшихся строк останется неизменной. Тогда программы, соответствующие этим строкам, будут выявлять те же неисправности, что и первоначальный тест. Таким образом, проблема минимизации первоначального теста может быть сведена к ряду исследовательских шагов по выбору минимальной функции покрытия.

При построении минимального контрольного теста (МКТ) исходим из следующих допущений:

1) в каждый фиксированный момент времени неисправным может оказаться только один элемент устройства (схемы);

2) неисправность элемента носит устойчивый характер.

Пусть изучается некоторое устройство (схема), у которого имеется n входных сигналов

$$x_1, x_2, \ldots, x_{n-1}, x_n$$

и m элементов

$$a_1, a_2, \ldots, a_{m-1}, a_m,$$

для которых построена матрица неисправностей.

Простейший тест для данного устройства — полный тест, т. е. последовательная проверка правильности работы на всех наборах входных переменных. Однако такой тест является избыточным, так как элементы проверяются многократно при различных входных наборах. В этом случае возникает задача определения такого минимального количества входных наборов, на которых каждый элемент проверяется не менее одного раза.

Множество наборов входных переменных G можно представить в следующем виде:

$$\begin{aligned}
&\bar{x}_1\bar{x}_2 \ldots \bar{x}_{n-1}\bar{x}_n = q_0\\
&\bar{x}_1\bar{x}_2 \ldots \bar{x}_{n-1}x_n = q_1\\
&\bar{x}_1\bar{x}_2 \ldots x_{n-1}\bar{x}_n = q_2\\
&\cdots\cdots\cdots\\
&x_1x_2 \ldots x_{n-1}\bar{x}_n = q_{2^n-2}\\
&x_1x_2 \ldots x_{n-1}x_n = q_{2^n-1}
\end{aligned}$$

Входные наборы множества G и элементы, проверяемые на этих наборах, свяжем некоторой функцией:

$$Bq_0 = A_0,\ Bq_1 = A_1,\ \ldots,\ Bq_{2^n-1} = A_{2^n-1},$$

где

$$A_0 \wedge A_1 \wedge A_2 \wedge \ldots \wedge A_{2^n-1} \subset A.$$

Подмножество элементов, проверяемых на наборе q_i, может быть записано так:

$$A_i = \{a_{i1},\ a_{i2},\ \ldots,\ a_{ik}\}.$$

Свойством подмножеств A_i является наличие как пересекающихся, так и непересекающихся подмножеств.

В этом случае задачу составления МКТ можно сформулировать следующим образом. В множестве G необходимо найти подмножество T наименьшей мощности, где для любого $a_j \in A$ найдется такое $q_i \in T$, что $a_j \in B\, q_i$.

В такой постановке данная задача сводится к задаче о минимальном покрытии таблицы и решается с помощью следующего алгоритма. Составляется булево произведение:

$$\Sigma q(a_1) \wedge \Sigma q(a_2) \wedge \ldots \wedge \Sigma q(a_m),$$

где

$$\Sigma q(a_i) = q_{i1} \vee q_{i2} \vee \ldots \vee q_{ik},$$

а подмножество наборов соответственно составляют:

$$Bq_{i1} = A_{i1},\ Bq_{i2} = A_{i2},\ \ldots,\ Bq_{ik} = A_{ik}.$$

Причем

$$A_{i1} \cap A_{i2} \cap \ldots \cap A_{ik} = A_i$$

и A_i состоит из одного элемента a_i. Другими словами, находим некоторую совокупность наборов

$$\{q_{i1}, q_{i2}, \ldots, q_{ik}\},$$

на которой проверяется элемент a_i.

Полученное выражение приводим к дизъюнктивной нормальной форме с последующим выбором слагаемого с минимальным количеством членов. Это слагаемое (одно или несколько) и будет минимальным контрольным тестом по определению.

Применяя рассмотренный алгоритм к матрице неисправностей (табл. 6.1), получим следующие минимальные наборы:

$$T_1 = \{q_1, q_2, q_7, q_{10}, q_{11}\};$$

$$T_2 = \{q_2, q_7, q_{11}, q_{12}, q_{13}\};$$

$$T_3 = \{q_1, q_2, q_7, q_{13}, q_{17}\}.$$

Таблица 6.1

Набор переменных	Программы испытаний (значения переменных)				Проверяемые элементы устройства (отмечены 1)														
	x_1	x_2	x_3	x_4	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{20}
q_1	0	0	0	1		1		1		1				1		1				1
q_2	0	0	1	0		1				1		1	1		1					
q_3	0	0	1	1				1												
q_4	0	1	0	0		1			1			1			1				1	
q_5	0	1	0	1								1								
q_6	0	1	1	0			1												1	
q_7	0	1	1	1			1		1	1			1				1		1	
q_{10}	1	0	0	0	1			1			1									1
q_{11}	1	0	0	1						1			1		1			1		
q_{12}	1	0	1	0		1								1						
q_{13}	1	0	1	1	1			1			1					1				1
q_{14}	1	1	0	0						1										
q_{15}	1	1	0	1	1		1					1					1			
q_{16}	1	1	1	0	1		1											1		1
q_{17}	1	1	1	1				1										1		

Построение упорядоченного контрольного теста

При конструировании МКТ не учитывались вероятности появления неисправностей, и вопрос о последовательности проверки устройства не ставился. Выходы из строя элементов устройства являются независимыми случайными событиями равной вероятности, так как элементы одного блока выполняются в основном из сходных деталей, имеющих одинаковый гарантийный срок службы.

Таким образом, если устройство, включающее m элементов, испытывается на наборе, проверяющем только m_i элементов, то вероятность обнаружения неисправного элемента составит:

$$P_i = \frac{m_i}{m}.$$

Среднее количество испытаний, которое необходимо выполнить до обнаружения неисправного элемента, равно:

$$M[s] = \sum_{i=1}^{n} ip_i,$$

где $M[s]$ — математическое ожидание среднего количества испытаний s;

n — длина минимального контрольного теста;

i — порядковый номер испытания (входного набора) в МКТ.

Упорядоченный контрольный тест характеризуется таким расположением наборов входных переменных, при котором величина $M[s]$ для данного контрольного теста минимальна.

Упрощенный алгоритм выбора упорядоченного контрольного теста по матрице неисправностей состоит в следующем.

Строка МКТ с наибольшим количеством единиц выбирается первой в упорядоченном тесте. Далее вычеркиваются единицы во всех столбцах, в которых находились единицы в ранее выбранной строке, и выбирается строка с максимальным количеством оставшихся единиц. Процедура поиска повторяется до тех пор, пока все наборы МКТ не будут выбраны. Например, упорядочение МКТ T_1 дает следующую последовательность применения наборов:

$$T_1' = \{q_1, q_7, q_2, q_{10}, q_{11}\}$$

и среднюю длину $M[s]=2.2$.

Для доказательства эффективности упорядочения достаточно, рассчитав величину $M[s]$ упорядоченного МКТ, сравнить ее с величиной среднего количества испытаний для обнаружения неисправности для любого неупорядоченного теста.

Построение минимального диагностического теста

Для практического применения более важными являются диагностические свойства таблицы. Пусть каким-то способом получена информация о том, правильно или неправильно выполняется каж-

дая программа теста. Поставим в соответствие каждому прогону теста на машине двоичное число по следующему правилу: k-й разряд этого числа равен 0, если k-я программа выполнилась правильно, и 1 — в противном случае. Это двоичное число в дальнейшем будем называть результатом теста.

Если в машине имеется одна и только одна из отмеченных в матрице неисправностей, то после выполнения теста в большинстве случаев будет получен результат, совпадающий со столбцом матрицы, соответствующим этой неисправности. Таким образом, матрица неисправностей может быть использована для определения с некоторой степенью точности местонахождения неисправности.

Требование к диагностическому тесту сводится к тому, чтобы в результате испытаний однозначно определить элемент, вышедший из строя. Базой для построения минимального диагностического теста является упорядоченный МКТ.

Основной принцип построения диагностического теста состоит в том, что для каждого шага испытания строится минимальный контрольный тест. Причем, если предыдущий результат правильный, то контрольный тест строится для непроверенных элементов. Если же результат предыдущего испытания оказался отрицательным, то необходимо строить контрольный тест для тех элементов, которые проверялись на предыдущем наборе входных переменных. Применительно к рассматриваемой матрице неисправностей упорядоченный МКТ имеет следующий вид:

$$T_1' = \{q_1,\ q_7,\ q_2,\ q_{10},\ q_{11}\}.$$

Вычеркиваем в табл. 6.1 все столбцы, у которых в строке q_1 стоят нули, все строки, в которых после вычеркивания столбцов нет ни

Таблица 6.2

Наборы переменных	Элементы устройства					
	a_2	a_4	a_6	a_{12}	a_{14}	a_{20}
q_2	1		1			
q_3		1				
q_4	1					
q_7			1			
q_{10}		1				1
q_{11}			1			
q_{12}	1			1		
q_{13}		1				1
q_{14}			1			
q_{17}		1				

Таблица 6.3

Наборы переменных	Элементы устройства		
	a_4	a_{14}	a_{20}
q_3	1		
q_{10}	1		1
q_{16}			1
q_{17}	1		

одной единицы, а также строку q_1. В результате получаем табл. 6.2, по которой строим МКТ для условного устройства с элементами a_2, a_4, a_6, a_{12}, a_{14}, a_{20} и упорядочиваем его. В результате имеем

$$T' = \{q_{13},\ q_{12},\ q_{14}\}.$$

Произведем аналогичные преобразования табл. 6.2, для чего вычеркиваем все столбцы, у которых в строке q_{13} стоят нули, и все строки, у которых нет ни одной единицы. В результате получим табл. 6.3, по которой строим МКТ для условного устройства с элементами a_4, a_{14}, a_{20} и упорядочиваем его. Эти преобразования выполняются до тех пор, пока остается один столбец и соответствующий ему элемент. Далее процедура повторяется для следующего набора МКТ (q_7), но предварительно необходимо вычеркнуть для набора q_1 все столбцы, для которых по строке q_1 имелись единицы в связи с тем, что на этом наборе неисправности не обнаружено.

Процесс преобразования матрицы неисправностей продолжается до последнего набора упорядоченного МКТ (q_{11}), после чего по результатам испытаний строится условный диагностический тест, вершинами нулевого уровня которого являются наборы упорядоченного МКТ. Очевидно, что при проверке устройства на первом наборе упорядоченного МКТ в случае выявления неисправных элементов последние необходимо заменить и после этого перейти ко второму набору. Преимущество поэтапного способа состоит в том, что таблицы получаются меньшими по объему и работа с ними упрощается.

6.3. Некоторые характеристики испытательных программ

Характеристики качества контроля

Для выполнения сравнительной оценки различных схем контроля и последующего выбора оптимальных вариантов рассмотрим некоторые характеристики качества контроля.

Обозначим через n число возможных ошибок в устройстве. При этом

$$n = \sum_{k=1}^{r} n_k, \tag{6.6}$$

где n_k — число возможных ошибок k-го элемента схемы;

r — число элементов в схеме (устройстве).

Пусть m — число ошибок, выявляемых испытательной программой. Тогда отношение

$$L = \frac{m}{n}, \tag{6.7}$$

называемое логической полнотой охвата схемы контроля, характеризует избирательность схемы контроля. Этот показатель представляет собой вероятность определения того элемента устройства, в виде ошибки которого проявилась физическая неисправность.

Режимная глубина контроля, осуществляемого принятой схемой реализации испытательной программы, оценивается отношением

$$R = \frac{s}{M}, \tag{6.8}$$

где s — число физических неисправностей, выявляемых испытательной программой;

M — число физических неисправностей, которые могут вызвать любую ошибку из числа m ошибок, определяемых испытательной программой.

Глубиной контроля оценивается вероятность обнаружения испытательной программой элемента, имеющего физическую неисправность, вызывающую хотя бы в одном из режимов ошибку из числа ошибок, обнаруживаемых в данной испытательной программе.

Обозначив через N число всех возможных физических неисправностей схемы, введем показатель эффективности контроля, выполняемого испытательной программой:

$$E = \frac{s}{N}. \tag{6.9}$$

Этим показателем оценивается вероятность обнаружения любой из всех возможных физических неисправностей схемы. При полном охвате контролем всех элементов схемы и полной глубине контроля эффективность контроля оценивается единицей.

Рассмотрим поведение функции эффективности контроля при фиксированном множестве ошибок, определяемых испытательной программой, и при расширении этого множества. Если множество ошибок фиксировано, то

$$L = \frac{m}{n} = \text{const}.$$

При этом $M = \text{const}$ и, следовательно,

$$\frac{M}{N} = \text{const}.$$

Из (6.8) определим

$$s = R \cdot M,$$

а из (6.9)

$$E = R\frac{M}{N}.$$

Следовательно, в рассматриваемом случае с увеличением R возрастает E. Таким образом, для фиксированного набора с целью повышения эффективности контроля необходимо выбрать испытательную программу, обеспечивающую большую величину R.

Если допустить расширение множества ошибок, определяемых испытательной программой, то эффективность контроля в этом случае не убывает, так как величина s либо сохраняет свое значение, либо возрастает.

Следовательно, показатели L, R и E являются характеристиками качества контроля, осуществляемого испытательной программой.

Основными командами испытательной программы являются такие команды, при исполнении которых работают элементы контролируемой схемы и по результату реализации которых выявляется неисправность. Все остальные команды испытательной программы являются вспомогательными.

Обозначим через t время выполнения основных команд испытательной программы, а через T — время выполнения всех ее команд. Тогда отношение

$$K = \frac{t}{T} \tag{6.10}$$

является коэффициентом полезного действия испытательной программы.

Для оценки испытательной программы по времени ее выполнения можно использовать характеристику

$$B = \frac{s}{T}, \tag{6.11}$$

называемую относительным быстродействием испытательной программы. Эта характеристика оценивает способность испытательной программы определять наличие различных физических неисправностей за единицу времени. Предполагается, что в исследуемом устройстве имеются все неисправности и испытательная программа, определяя каждую из них, не останавливается, а продолжает контроль. На основании зависимостей (6.9) и (6.11) получаем

$$B = E\frac{N}{T}. \tag{6.12}$$

Выполнив замену T на основании (6.10), имеем

$$B = EK\frac{N}{t}. \tag{6.13}$$

Из полученной зависимости следует, что относительное быстродействие связывает основные характеристики качества контроля — эффективность и коэффициент полезного действия.

Одной из основных характеристик качества испытательной программы является ее надежность. Обозначим через b число неисправностей, не влияющих на работу вспомогательных команд испытательной программы и на работу тех элементов машины, которые функционируют как при выполнении основных, так и при выполнении вспомогательных команд и не принадлежат к контролируемой схеме. Тогда надежность работы испытательной программы выражается отношением

$$H = \frac{b}{N} \tag{6.14}$$

и представляет собой вероятность того, что имеющаяся неисправ-

ность не повлияет на выполнение вспомогательных команд испытательной программы и на работу тех элементов, которые функционируют как при выполнении основных, так и при выполнении вспомогательных команд, но не принадлежат к контролируемой схеме.

Надежность испытательной программы всегда меньше единицы. Для повышения надежности применяется ряд приемов, основные среди которых: создание таких режимов работы для вспомогательных команд, при которых вероятность проявления неисправности минимальна; уменьшение количества вспомогательных команд в испытательной программе.

Для повышения надежности выполнения испытательной программы применяют комплексы таких программ. Предположим, что испытательная программа выполняется после некоторого набора других испытательных программ и после устранения выявленных ими неисправностей. Если уже устранено b_1 неисправностей, то в испытуемой схеме остается обнаружить $b'=b-b_1$ неисправностей, которые не влияют на выполнение вспомогательных команд и на работу элементов машины, функционирующих при выполнении основных команд, но не входящих в контрольную схему, при условии выполнения всех испытательных программ, предшествующих данной, и устранения неисправностей, обнаруженных этими программами.

Условной надежностью выполнения испытательной программы называется отношение

$$H'=\frac{b'}{N}. \qquad (6.15)$$

Исходя из этой характеристики можно ввести определение системы испытательных программ. *Системой (комплексом) испытательных программ,* служащих для наладки и контроля машины, называется некоторый набор программ, составленный так, что если выполнять программы из этого набора в определенном порядке, заданном при построении системы, и устранять неисправности по мере их обнаружения, то каждая из программ последовательности будет иметь достаточно высокую условную надежность выполнения. Проверка всей машины с помощью системы испытательных программ должна выполняться за приемлемое время.

В практике обычно имеют два комплекса испытательных программ. Первый комплекс программ является наладочным, имеет свои специфические особенности, используется при первоначальной наладке машин, а также при их наладке после годовой и полугодовой профилактики. Второй комплекс программ предназначен для проведения контроля и поиска неисправностей в текущей эксплуатации машины. Любой комплекс испытательных программ включает как контрольные, так и диагностические программы (тесты).

Контрольные тесты реализуют специальные проверочные алгоритмы, у которых можно отметить следующие общие черты:

1) проверка выполняется путем сравнения результатов, полученных при реализации алгоритма, с эталонными;

2) окончательным итогом работы алгоритма является утверждение о наличии или отсутствии неисправностей в проверяемых схемах;

3) кроме окончательного результата проверки, выдается протокольное сообщение о ходе контрольного теста.

С целью увеличения условной надежности контрольных тестов их конструируют так, чтобы соблюдался принцип расширяющихся областей, сущность которого состоит в следующем:

1) ранее проверенные элементы используются для целей контроля, т. е. элементы, проверяемые какой-либо предыдущей испытательной программой, в последующих программах могут быть использованы для выполнения вспомогательных команд. При этом допустимый режим их работы определяется глубиной контроля той программы, в которой их работоспособность проверялась;

2) ранее проверенные элементы контролируются с большей глубиной и в более тяжелых режимах;

3) в очередном контрольном тесте включаются в контроль новые группы элементов.

Входной информацией для диагностического теста служит факт наличия неисправности, выявленной контрольным тестом, и некоторая дополнительная информация, образующаяся в результате работы контрольного теста. Диагностический тест строится по принципу сужения области контроля. В диагностическом тесте постепенно сужается круг элементов, проверяемых на наличие неисправности.

Характеристики качества контроля и функционирования контрольных и диагностических программ

На базе ранее введенных показателей качества контроля испытательных программ уточним характеристики контрольных и диагностических тестов.

Для контрольных тестов полнота охвата контролем определяется отношением

$$L_k = \frac{m_k}{n} \tag{6.16}$$

и показывает вероятность того, что если в контролируемой схеме имеется неисправность, проявляющаяся в виде ошибки какого-либо элемента, то эта ошибка проявляется хотя бы в одном из возможных режимов работы контрольного теста. Величина $m_k (m_k \geqslant m)$ представляет собой количество ошибок, факт наличия которых определяется контрольным тестом.

Глубина контроля контрольного теста определяется отношением

$$R_k = \frac{s_k}{M_k} \tag{6.17}$$

и представляет собой вероятность установления факта наличия неисправности. В этом выражении $s_k (s_k \geqslant s)$ — число неисправностей,

которые определяются контрольным тестом, а M_k — число тех неисправностей, которые могут вызвать хотя бы в одном из режимов ошибку из числа выявляемых данным контрольным тестом.

Эффективность контроля определяется отношением

$$E_k = \frac{s_k}{N} \tag{6.18}$$

и представляет собой вероятность обнаружения любой из возможных неисправностей машины.

Полнота охвата диагностического теста определяется отношением

$$L_q = \frac{m}{m_k} \tag{6.19}$$

и представляет собой вероятность того, что если контрольный тест определил факт наличия неисправности из числа m_k, то диагностический тест определит элемент, в виде ошибки которого проявилась физическая неисправность схемы.

Глубина контроля диагностического теста определяется отношением

$$R_q = \frac{s}{M_q} \tag{6.20}$$

и представляет собой вероятность выявления неисправного элемента. В зависимости (6.20) через M_q обозначено число всех тех неисправностей, которые могут вызвать любую ошибку из числа m.

Эффективность диагностического теста определяется отношением

$$E_q = \frac{s}{s_k} \tag{6.21}$$

и представляет собой вероятность локализации места любой неисправности, обнаруженной с помощью контрольного теста.

Для сравнительной оценки различных контрольных и диагностических тестов необходимо ввести характеристики их качества.

Обозначим через t_k и t_q соответственно время выполнения основных команд контрольного и диагностического тестов, а через T_k и T_q — время выполнения этих процедур. Тогда коэффициент полезного действия рассматриваемых процедур может быть записан так:

$$K_k = \frac{t_k}{T_k}; \tag{6.22}$$

$$K_q = \frac{t_q}{T_q}. \tag{6.23}$$

В силу последовательного исполнения этих процедур можно записать:

$$T \approx T_k + T_q;$$
$$t \approx t_k + t_q.$$

Поэтому коэффициент полезного действия всей испытательной программы можно записать в виде следующего соотношения

$$K = \frac{t}{T} = \frac{t_k + t_q}{T_k + T_q}. \qquad (6.24)$$

Полученное соотношение справедливо в случае обнаружения контрольным тестом факта наличия неисправности с последующей локализацией ее места диагностическим тестом. Контрольные тесты работают, как правило, в различных режимах профилактического контроля схемы с целью повышения надежности выявления неисправностей и их своевременного прогнозирования. Для обнаружения неисправности контрольный тест выполняется на всех режимах профилактического контроля и может передать управление диагностическому тесту по мере обнаружения неисправности. По максимальной оценке (наличие P режимов профилактического контроля) зависимость (6.24) может быть уточнена:

$$K = \frac{Pt_k + st_q}{PT_k + sT_q}. \qquad (6.25)$$

Важное значение для совершенствования конструкций контрольных и диагностических процедур с целью повышения показателей качества их работы имеет детально организованный сбор статистических данных о выходе из строя отдельных элементов машин в процессе их эксплуатации. Накопление и последующий анализ статистического материала по фактам выхода из строя отдельных элементов и по характеру работы испытательных программ в процессе их эксплуатации обеспечит дальнейшее совершенствование алгоритмов контроля и диагностики вычислительных машин.

ВОПРОСЫ К ГЛАВЕ

1. Назначение испытательных программ.
2. Методы конструирования испытательных программ.
3. Краткая характеристика метода построения минимального теста.
4. Краткая характеристика метода построения упорядоченного минимального контрольного теста.
5. Краткая характеристика метода построения минимального диагностического теста.
6. Перечислить характеристики качества контроля.
7. Определить логическую полноту охвата схемы контроля.
8. Определить режимную глубину контроля.
9. Определить эффективность контроля.
10. Перечислить характеристики качества испытательных программ.
11. Перечислить характеристики качества контроля и функционирования контрольных и диагностических программ.

Глава 7
КОНТРОЛЬНЫЕ ЗАДАЧИ

7.1. Оценка надежности ЭВМ

Надежность систем обработки информации имеет специфические особенности по сравнению с надежностью электронной аппаратуры вообще. Термин «надежность» можно определить как способность машины безошибочно выполнять алгоритм переработки информации, поступающей на ее входы, в течение определенного времени при заданных условиях эксплуатации.

Возможность своевременного решения задачи определяется ее временными характеристиками: временем, необходимым непосредственно для решения задачи на машине при отсутствии отказов, общим временем решения задачи и резервом машинного времени.

Под общим временем решения задачи понимается промежуток времени между моментом, к которому должны быть подготовлены исходные данные для решения задачи, и запланированным сроком выдачи результатов. Резерв машинного времени определяется разностью между общим временем на решение задачи и собственно временем решения задачи без учета возможных отказов. Это время используется для устранения отказов и их последствий, т. е. оно представляет собой временную избыточность, которая в определенной степени снижает отрицательное воздействие отказов на результат функционирования ЭВМ по решению задач.

Таким образом, надежность ЭВМ определяется не только параметрами надежности, но и задачами, которые лимитируют количество возможных отказов исходя из имеющегося резерва машинного времени, необходимого для их устранения.

Для оценки надежности необходимо найти зависимость между своевременностью решения задачи, параметрами надежности и резервом машинного времени. Получение такой функциональной зависимости дает возможность определения вероятности своевременного решения задачи при известных параметрах надежности и заданном резерве машинного времени, резерва машинного времени, необходимого для своевременного решения задачи с заданной вероятностью при известных параметрах надежности ЭВМ. Кроме того, можно определить требуемые параметры надежности ЭВМ при заданной вероятности своевременного решения конкретных задач и известных резервах машинного времени.

Решение этих задач позволяет обосновать требования по надежности на проектируемую ЭВМ или выбрать наиболее подходящий в отношении надежности тип ЭВМ.

Сформулируем основные ограничения, накладываемые на модель зависимости между своевременностью решения задачи и параметрами надежности.

Предполагаем, что время решения задачи t в однопрограммном режиме заранее известно и постоянно. Потоки отказов и устране-

ний отказов принимаем пуассоновскими. Интенсивность потока устранения отказов для задач определенного типа считаем постоянной и равной μ. Время устранения отказов включает время поиска места неисправности, время устранения ее, время проверки ЭВМ и время восстановления утраченной или искаженной информации. При этом полагаем, что время устранения дополнительных отказов, появляющихся в процессе устранения имеющегося отказа, входит во время устранения этого отказа.

При установлении зависимости между вероятностью своевременного решения задачи и параметрами надежности λ и μ для того, чтобы математическая модель лучше соответствовала действительному положению вещей, необходимо отказаться от обычно принимаемого допущения о том, что интенсивность отказов постоянна и одинакова при решении всех задач. Это вызвано тем обстоятельством, что практически интенсивность отказов λ меняется от задачи к задаче и даже в процессе решения одной задачи.

Чтобы отразить изменение интенсивности отказов в процессе решения задачи, целесообразно разбить решение на последовательные этапы, различающиеся относительно постоянным составом используемых устройств. Такими этапами могут быть ввод исходных данных в память ЭВМ, выполнение счета по заданному алгоритму и вывод результатов решения задачи.

Интенсивность отказов ЭВМ на каждом этапе решения задач с учетом вида используемых устройств можно получить как сумму интенсивностей отказов устройств, используемых постоянно, т. е. для всех типов задач, и устройств, используемых для данного типа задачи:

$$\lambda_{ij} = \sum_{i=1}^{k} \lambda_{ni} + \sum_{i=1}^{l} \lambda_{nij},$$

где λ_{ij} — суммарная интенсивность отказов на i-м этапе при решении задач j-го типа;

$\sum_{i=1}^{k} \lambda_{ni}$ — сумма интенсивностей отказов k устройств, постоянно задействованных на i-м этапе;

$\sum_{i=1}^{l} \lambda_{nij}$ — сумма интенсивностей отказов остальных устройств, задействованных на i-м этапе для решения j-й задачи.

Время решения задачи на каждом этапе можно с известной степенью точности определить исходя из опытных данных или на основе анализа алгоритма задачи.

Следующим шагом при оценке вероятности своевременного решения задачи является определение вероятности появления n отказов при решении задачи в три этапа. Она равна сумме всех комбинаций произведений вероятностей появления какого-то, меньшего или равного n количества возможных отказов на каждом этапе. При этом комбинации представляют собой такие различные соче-

тания количеств отказов на отдельных этапах, что сумма отказов на всех этапах равна n.

Вероятность появления n отказов при решении задачи в три этапа можно получить по формулам:

$$P(n) = \sum_{\substack{n=\eta+p+q \\ 0\leqslant\eta,\ p,\ q\leqslant n}} P_1(\eta)\, P_2(p)\, P_3(q),$$

где $P_1(\eta)$ — вероятность появления η отказов на первом этапе;
$P_2(p)$ — вероятность появления p отказов на втором этапе;
$P_3(q)$ — вероятность появления q отказов на третьем этапе;
$\sum_{n=\eta+p+q}$ — сумма всех комбинаций, которые можно образовать из возможных количеств на отдельных этапах так, чтобы сумма отказов оказалась равной n.

Для простейшего потока отказов эта формула примет следующий вид:

$$P_j(n) = \sum_n \frac{(\lambda_{1j}t_1)^\eta}{\eta!} \times e^{-\lambda_{1j}t_1} \times \frac{(\lambda_{2j}t_2)^p}{p!} e^{-\lambda_{2j}t_2} \times \frac{(\lambda_{3j}t_3)^q}{q!} e^{-\lambda_{3j}t_3}$$

Для упрощения полученной зависимости преобразуем параметр простейшего потока:

$$\lambda_{ij}t_i = \lambda_{ij}\frac{t_i}{t}\,t = \lambda_{yij}t,$$

где t — полное время решения задачи в три этапа;
λ_{yij} — условная интенсивность появления отказов на i-м этапе.

Введем обозначения условных интенсивностей появления отказов в формулу вероятности появления отказов и преобразуем ее:

$$P(n) = \Sigma\, \frac{(\lambda_{y1j}t)^\eta}{\eta!} e^{-\lambda_{y1j}t} \cdot$$

$$\cdot \frac{(\lambda_{y2j}t)^p}{p!} e^{-\lambda_{y2j}t} \cdot \frac{(\lambda_{y3j}t)^q}{q!} e^{-\lambda_{y3j}t} = e^{-t\sum_{i=1}^{3}\lambda_{yij}} \sum_{n=\eta+p+q} \frac{t^n \lambda_{y1j}^\eta \lambda_{y2j}^p \lambda_{y3j}^q}{\eta!\ p!\ q!}.$$

Рассмотрим, чему равна вероятность получения 0, 1, 2 отказов по этой формуле. Вероятность безотказной работы ЭВМ ($n=0$) при решении задачи:

$$P(0) = e^{-t\sum_{i=1}^{3}\lambda_{yi}}.$$

Вероятность появления одного отказа при решении задачи:

$$P(1) = t\sum_{i=1}^{3}\lambda_{yi}e^{-t\sum_{i=1}^{3}\lambda_{yi}}.$$

Вероятность появления двух отказов при решении задачи:

$$P(2) = \frac{\left(t\sum_{i=1}^{3}\lambda_{yi}\right)^2}{2!} e^{-t\sum_{i=1}^{3}\lambda_{yi}}.$$

Обозначив сумму условных интенсивностей появления отказов на всех этапах через Λ получим, что вероятность появления n отказов в процессе решения задачи можно определить по формуле Пуассона с параметром Λt:

$$P(n) = \frac{(\Lambda t)^n}{n!} e^{-\Lambda t}.$$

На основании полученных данных о характере задачи и параметрах надежности ЭВМ определим вероятность своевременного решения отдельной задачи. Физический смысл своевременного решения задачи состоит в том, что суммарное время устранения отказов, появившихся во время решения задачи, должно быть меньше или равно резерву машинного времени.

При определении вероятности своевременного решения задачи принимаем следующие обозначения:

t — время, необходимое для решения задачи при отсутствии отказов;

τ — суммарная длительность устранения отказов и их последствий в процессе решения задачи;

T_Δ — резерв машинного времени для данной задачи;

$P_k(t)$ — вероятность того, что за время t решения задачи произошло k отказов;

$W_k(\tau \leqslant T_\Delta)$ — вероятность того, что при k отказах суммарное время устранения неисправностей не превысило резерва времени;

$W(\tau \leqslant T_\Delta)$ — вероятность того, что суммарное время устранения отказов и их последствий в процессе решения задачи меньше допустимого времени T_Δ.

По формуле полной вероятности получим выражение для оценки вероятности своевременного решения задачи:

$$W(\tau \leqslant T_\Delta) = \sum_{k=0}^{\infty} P_k(t)\, W_k(\tau \leqslant T_\Delta).$$

Очевидно, что при $k=0$ $W_k(\tau \leqslant T_\Delta) = 1$, следовательно, предыдущее равенство можно переписать так:

$$W(\tau \leqslant T_\Delta) = \sum_{k=1}^{\infty} P_k(t)\, W_k(\tau \leqslant T_\Delta) + P_0(t).$$

Так как поток устранения отказов ЭВМ принят простейшим, то вероятность выполнения неравенства $\tau \leqslant T_\Delta$ для целых положительных k равна:

$$W_k(\tau \leqslant T_\Delta) = \sum_{s=k}^{\infty} \frac{(\mu \cdot T_\Delta)^s}{s!} \cdot e^{-\mu T_\Delta} = 1 - \sum_{s=0}^{k-1} \frac{(\mu \cdot T_\Delta)^s}{s!} \cdot e^{-\mu T \Lambda}.$$

Вероятность появления k отказов в течение времени решения задачи t в соответствии с принятыми допущениями определяется по формуле

$$P_k(t) = \frac{(\Lambda t)^k}{k!} e^{-\Lambda t}.$$

Подставив выражение для $P_k(t)$ и $W_k(\tau \leqslant T_\Delta)$ в формулу вероятности своевременного решения задачи, после ряда упрощений получим:

$$W(\tau \leqslant T_\Delta) = 1 - e^{-(\mu T_\Delta + \Delta t)} \sum_{k=1}^{\infty} \left(\frac{(\Delta t)^k}{k!} \cdot \sum_{s=0}^{k-1} \frac{(\mu T_\Delta)^s}{s!} \right).$$

Если известно, что начиная с $(n+1)$ члена бесконечного ряда все последующие члены этого ряда не вносят существенных изменений в результат, полученный с учетом первых n членов ряда, то можно ограничить бесконечный ряд n-м членом и получить более удобную для вычислений формулу

$$W(\tau \leqslant T_\Delta) = 1 - e^{-(\mu T_\Delta + \Delta t)} \sum_{k=1}^{n} \left(\frac{(\Delta t)^k}{k!} \sum_{s=0}^{k-1} \frac{(\mu T_\Delta)^s}{s!} \right).$$

Полученная формула представляет собой функциональную зависимость вероятности своевременного решения задачи от величины параметров надежности и резерва машинного времени.

Анализ этой зависимости показывает, что резерв машинного времени связан со средним временем устранения отказа обратной зависимостью. Это позволяет недостаточно высокую интенсивность восстановления отказов компенсировать увеличением ресурсов машинного времени.

Зная оценку надежности функционирования ЭВМ, можно подобрать в качестве контрольных задач такие алгоритмы обработки информации, по которым обеспечивалась бы разносторонняя комплексная проверка работоспособности ЭВМ.

7.2. Программы комплексной проверки ЭВМ

В качестве испытательных программ, выполняющих комплексную проверку машины, могут использоваться также программы некоторых конкретных задач, называемых контрольными задачами. Комплексная проверка работоспособности машины обычно проводится на заключительной стадии наладки при пуске ЭВМ или после завершения предусмотренного объема профилактических работ полугодовой и годовой профилактики.

В соответствии с математической моделью определения надежности своевременного решения задачи в контрольной задаче должны присутствовать три этапа. Алгоритм задачи должен охватывать как можно большее число устройств, входящих в комплект машины. Естественно, что в одной контрольной задаче это сделать не удается и, как правило, в программное обеспечение машины включается несколько контрольных задач.

Контрольные задачи предварительно решаются с целью получения результатов решения, которые впоследствии используются для сопоставления с результатами частных решений. Совпадение результатов подтверждает в определенной мере работоспособность ЭВМ. При несовпадении результатов производится пропуск контрольных, а затем и диагностических тестов с целью выявления неисправностей.

Применительно к ЭВМ «Минск-32» рассмотрим некоторые алгоритмы контрольных задач.

Алгоритм решения системы дифференциальных уравнений обеспечивает комплексную проверку правильности функционирования вычислителя совместно с устройством широкой печати. Алгоритм предусматривает решение шести систем дифференциальных уравнений второго порядка, построенных по следующему принципу:

а) исходная система состоит из восьми уравнений:

$$1)\ \frac{\partial v_x}{\partial t} = f_1 = -w_z v_y - 0.0001771v^2 + 0.01294\alpha^2 v^2 + 0.0007692v^2 - qy;$$

$$2)\ \frac{\partial v_y}{\partial t} = f_2 = -w_z v_y + 0.02142\alpha v^2 + 0.000769v^2 - qx;$$

$$3)\ \frac{\partial w_z}{\partial t} = -0.0004432\alpha v^2 - 0.0002044\delta v^2 - 0.02374 w_z v + 0.01187 \frac{v}{v_x}(f_2 + \alpha f_1);$$

$$4)\ \frac{\partial \delta}{\partial t} = -11.44\delta + 0.56\alpha + 4.8w_z + 7.64;$$

$$5)\ \frac{\partial \theta}{\partial t} = w_z;$$

$$6)\ \frac{\partial y}{\partial t} = xw_z;$$

$$7)\ \frac{\partial x}{\partial t} = -yw_z;$$

$$8)\ \frac{\partial v}{\partial t} = \frac{v_x f_1 + v_y f_2}{v};$$

б) к системе добавляется уравнение $\frac{dt}{dt} = 1$,

$$D = -\frac{v_y}{v_x}.$$

Каждая из систем решается методами Рунге-Кутта и Адамса. Вычисления производятся в 2900 точках при постоянном шаге интегрирования $h=0.0125 (1.5 \leqslant t < 38)$.

Для вычисления по методу Рунге-Кутта используется следующая зависимость:

$$y_{k+1,\ i} = y_{k,\ i} + \sum_{j=1}^{4} \left(\frac{1}{6} M_{j+1} h k_{ji}\right),$$

где

$$k_{1i} = f_i (y_{k1},\ y_{k2},\ \ldots,\ y_{kn});$$

$$k_{2i} = fi \left(y_{k1} + \frac{M_1 h}{2};\ y_{k2} + \frac{M_1 h k_{11}}{2};\ \ldots;\ y_{kn} + \frac{M_1 h k_{1n}}{2}\right);$$

$$k_{3i} = fi \left(y_{k1} + \frac{M_2 h}{2};\ y_{k2} + \frac{M_2 h k_{21}}{2};\ \ldots;\ y_{kn} + \frac{M_2 h k_{2n}}{2}\right);$$

$$k_{4i} = fi \left(y_{k1} + \frac{M_3 h}{2};\ y_{k2} + \frac{M_3 h k_{31}}{2};\ \ldots;\ y_{kn} + \frac{M_3 h k_{3n}}{2}\right);$$

$$M_1 = 1;\ M_2 = 1;\ M_3 = 2;\ M_4 = 2;\ M_5 = 1;$$

$$i = 1,\ 2,\ \ldots,\ 9.$$

Для вычисления по методу Адамса используют:

а) экстраполяционную формулу Адамса:

$$y_{m+1,\ i} = y_{m,\ i} + \frac{h}{1440} (7244 y'_{mi} - 7923 y'_{m-1,\ i} + 9982 y'_{m-2,\ i} -$$

$$- 7298 y'_{m-3,\ i} + 2877 y'_{m-4,\ i} - 475 y'_{m-5,\ i});$$

б) интерполяционную формулу Адамса:

$$y_{m+1,\ i} = y_{mi} + \frac{h}{1440} (421 y'_{m+1,\ i} + 1697 y'_{mi} - 1338 y'_{m-1,\ i} +$$

$$+ 1022 y'_{m-2,\ i} - 443 y'_{m-3,\ i} + 81 y'_{m-4,\ i});$$

$$i = 1,\ 2,\ \ldots,\ 9.$$

При решении системы уравнений методом Рунге-Кутта через каждые 100 точек происходит запоминание значений аргумента t и функций.

При решении этих систем производится вывод на устройство широкой печати ранее запоминаемых значений функций в виде графиков, в которых на каждую из строк отводится 56 позиций.

Время решения всей задачи из шести систем составляет около 50 мин. Решение одной системы осуществляется за 8 мин. Промежуток времени между печатью на УПч в методе Рунге-Кутта составляет 14 сек. Промежуток времени между печатью на УПч в методе Адамса составляет 33 сек.

Перед решением каждой системы печатается ее номер. Через каждые 290 точек на печать выдаются значения аргумента t и функций v_x, v_y, w_z, δ, Θ, y, x, v. Для каждой из систем печатается десять строк значений для метода Рунге-Кутта и десять строк для метода Адамса.

Контроль правильности решения производится сравнением полученных значений с контрольными результатами.

Алгоритм второй контрольной задачи предусматривает обращение матрицы сотого порядка. После выполнения обращения производится перемножение исходной и обратной матриц. Контрольная задача предназначена для комплексной проверки правильности функционирования вычислителя совместно с устройством вывода на широкую печать и устройством магнитной ленты.

Обращение матрицы производится методом разбиения на клетки. Расчет выполняется по формуле

$$A_k^{-1} = A_{k-1}^{-1} - \frac{A_{k-1}^{-1} U_k V_k A_{k-1}^{-1}}{1 - V_k A_{k-1} U_k},$$

где

$$k = 1, 2, \ldots, 99, 100;$$

$$V_k = \left(000 \ldots \frac{1}{K} \ldots 0\right);$$

$$A = \begin{vmatrix} \alpha_{11} & \alpha_{12} & \ldots & \alpha_{1k} & 0 \ldots 0 \\ \alpha_{21} & \alpha_{22} & \ldots & \alpha_{2k} & 0 \ldots 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \alpha_{k1} & \alpha_{k2} & \ldots & \alpha_{kk} & 0 \ldots 0 \\ \alpha_{k+1,\,1} & \alpha_{k+1,\,2} & \ldots & \alpha_{k+1,\,k} & 0 \ldots 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \alpha_{n1} & \alpha_{n2} & \ldots & \alpha_{nk} & 0 \ldots 0 \end{vmatrix};$$

$$U_k = \begin{vmatrix} \alpha_{1k} \\ \cdot \\ \cdot \\ \cdot \\ \alpha_{k-1,\,k} \\ \cdot \\ \cdot \\ \cdot \\ \alpha_{nk} \end{vmatrix}; \qquad A_0^{-1} = \begin{vmatrix} \frac{1}{\alpha_{11}} & 0 & 0 & \ldots & 0 \\ 0 & \frac{1}{\alpha_{22}} & 0 & \ldots & 0 \\ 0 & 0 & \frac{1}{\alpha_{33}} & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & \frac{1}{\alpha_{nn}} \end{vmatrix},$$

α_{ij} — элементы исходной матрицы $A(j, i=1, 2, \ldots, 99, 100)$;
A_k^{-1} — k-е приближение матрицы A^{-1}.

Вычисления производятся до сотого получения значения обратной матрицы. Элементы единичной матрицы

$$E = A \cdot A^{-1}$$

подсчитываются по формуле

$$l_{ij} = \alpha_{i1} \cdot \bar{\alpha}_{1j} + \alpha_{i2} \cdot \bar{\alpha}_{2j} + \ldots + \alpha_{i,\,100}\bar{\alpha}_{100,\,j},$$

где l_{ij} — элемент единичной матрицы;

$\bar{\alpha}_{kj}$ — элемент обратной матрицы.

Время решения задачи составляет около 66 мин. При этом вычислитель занят всего 16 мин. Для решения задачи используются 8К оперативной памяти, три лентопротяжных механизма и устройство широкой печати. За время одной итерации выполняется девяносто обращений к МЛ и выдается на широкую печать одна строка контрольных чисел. Контроль правильности решения задачи проверяется путем сравнения результатов вычислений с контрольными результатами.

ВОПРОСЫ К ГЛАВЕ

1. Что включается в понятие надежности ЭВМ?
2. Какова зависимость между надежностью и процессом алгоритмизации задач?
3. Какова зависимость между вероятностью своевременного решения задачи и параметрами надежности?
4. Определите вероятность появления n отказов при решении задачи в три этапа.
5. Запишите зависимость вероятности безотказной работы машины при решении задачи от интенсивности появления отказов.
6. От каких факторов зависит вероятность своевременного решения задачи?
7. Назначение и сущность проверки машины с помощью контрольных задач.
8. Режим работы первой контрольной задачи и перечень проверяемых устройств.
9. Режим работы второй контрольной задачи и перечень проверяемых устройств.

ЛИТЕРАТУРА

Диагностика неисправности вычислительных машин. Под ред. Н. В. Паутина. М., «Наука», 1965.

Миронов Г. А. Испытательные программы для контроля электронных цифровых машин. М., «Наука», 1964.

Современное программирование. Языки для экономических расчетов. Сборник статей № 2. М., «Советское радио», 1967.

Теория надежности и массовое обслуживание. Под ред. Б. В. Гнеденко. М., «Наука», 1969.

Чжен Г., Мэннинг Е., Метц Г. Диагностика отказов цифровых вычислительных систем. М., «Мир», 1972.

РАЗДЕЛ 3

СИСТЕМЫ ПРОГРАММИРОВАНИЯ

Глава 8

ХАРАКТЕРИСТИКА СИСТЕМ ПРОГРАММИРОВАНИЯ

8.1. Классификация систем программирования

В соответствии со структурой, уровнем формализации входного языка и целевым назначением систем программирования может быть принята следующая их классификация: машинные системы, процедурные системы, проблемные системы, вспомогательные системы.

Машинные системы программирования имеют входной язык, связанный с определенной машиной или семейством машин, и не могут быть приспособлены к иному виду машин, чем тот, для которого они разработаны. Наиболее типичными представителями такой системы программирования являются системы символического кодирования, автокоды, макрогенераторы.

Системы символического кодирования — одни из первых систем, созданных для автоматизации программирования с использованием входных языков уровня «один к одному». Этот уровень предполагает, что одной символической инструкции соответствует одна машинная команда или константа. Смысл применения данной системы состоит в использовании символической нотации вместо машинной, в применении автоматического распределения памяти и присвоения действительных адресов.

Системы символического кодирования являются базой для создания более совершенных систем автоматизации программирования по следующим причинам:

входной язык высшего уровня транслируется в символический, а машинная программа получается затем в результате работы транслятора с языка символического кодирования;

если ЭВМ имеет специфические устройства, которые не могут быть учтены в процедурной системе с языка высшего уровня, то целесообразно перейти от процедурного языка к символическому, чтобы эффективнее использовать эти особенности машины.

Кроме этого, система символического кодирования имеет и самостоятельное значение как система программирования малых ЭВМ, для которых нельзя создать транслятор с языков высшего уровня в силу ограниченных возможностей машин.

Система символического кодирования включает следующие составные части:

входной язык символического кодирования для записи программ и изменений к ним в удобной и наглядной форме;

транслятор, обеспечивающий контроль и перевод исходной программы в машинную программу;

библиотеку стандартных подпрограмм на машинном языке;

обслуживающие программы системы;

описания языка и инструкции по пользованию системой символического кодирования и порядку проведения работ по программированию, перфорации, трансляции и т. д.

Являясь машинно-ориентированным, язык символического кодирования требует от программиста знания основных приемов непосредственного программирования и позволяет ему в полной мере проявить искусство для написания эффективных программ.

Дальнейшим развитием автоматизации программирования на уровне машинно-ориентированных систем является использование автокодов. Входным языком автокодов является язык уровня «один к нескольким», т. е. одной автокодовой инструкции соответствует несколько машинных команд. Основу автокодового языка составляют макрокоманды, обозначающие функцию или процедуру одной записью. Макрокоманды переводятся в машинные команды одним из двух машинных путей: подстановкой или генерированием.

В подстановочной системе каждой макрокоманде соответствуют библиотечные модули на машинном языке, реализующие действия, предусматриваемые макрокомандой. Для настройки модулей на конкретную работу в макрокоманде должны быть указаны параметры очередного обращения, по которым транслятор выполняет настройку.

Генерирующая система содержит специальные программы, анализирующие макрокоманды. При проведении анализа выясняется, какую функцию требует исполнить программист, и формируются команды, реализующие эти функции.

Подстановочные и генерирующие системы содержат транслятор с символического языка. Поэтому программист при составлении программ может одновременно употреблять как символические команды, так и макрокоманды.

По сравнению с системой символического кодирования автокоды имеют следующие преимущества:

улучшение внутренней организации в целях ускорения трансляции;

наличие разнообразных макрокоманд вывода, обеспечивающих вывод на большинство внешних устройств в форме, пригодной для последующего ввода;

уменьшение трудовых затрат на составление программ за счет применения макрокоманд;

применение более совершенных методов обнаружения ошибок в исходных программах на этапе трансляции;

сокращение времени прохождения задач от их постановки до машинной реализации.

Макроязык конструируется по типу системы команд конкретной ЭВМ и поэтому лишен универсальности, свойственной языкам высшего уровня. В силу этого обстоятельства, а также из-за приспособленности к логической структуре определенной ЭВМ возможно построение более простых и компактных трансляторов с макроязыков автокодов. Программы, составленные с помощью автокодов, по своему качеству в большей степени приближаются к программам, составленным вручную программистом средней квалификации, отличаясь от последних не более чем на 10—15% по объему и времени решения задач.

В отличие от машинно-ориентированных систем программирования процедурные системы используют в качестве входного языка для описания алгоритмов задач различные алгоритмические языки, не зависящие от конкретных вычислительных машин. Эти языки, относящиеся к уровню «один к нескольким», выгодно отличаются от автокодов тем, что полностью освобождают программиста от записи машинных программ.

Основными компонентами процедурных систем программирования являются входной алгоритмический язык и компилятор с него на уровень систем символическоро кодирования или на уровень непосредственно машинных команд. В соответствии с целевым назначением алгоритмических языков можно рассматривать системы программирования, ориентированные на описание и обработку программ информации научно-технических и инженерных задач (АЛГОЛ, ФОРТРАН и др.), экономических задач (КОБОЛ, АЛГЭК, АЛГЭМ и др.), символической информации (ЛИСП, ЭПСИЛОН и др.), моделирования (СИМУЛА, СИМСКРИПТ) и другие системы, базирующиеся на конкретных алгоритмических языках.

Применение процедурных систем программирования позволяет в значительной мере сократить время прохождения задач от момента постановки до машинной реализации, обеспечивая повышение производительности труда программистов. Однако время счета по программам, полученным в результате трансляции, в определенной мере зависит от качества компилятора и даже в лучших разработках значительно превышает время счета по программам, полученным с использованием систем символического кодирования или автокодов.

В нашей стране наиболее распространенными процедурными системами являются системы, использующие алгоритмические языки АЛГОЛ, КОБОЛ, ФОРТРАН и АЛГЭК.

Проблемные системы программирования исключают какую-либо

работу программиста по разработке алгоритмов решаемых задач. Эти системы ориентированы на узкий класс однотипных задач. Примерами языков в таких системах являются специализированные инженерные языки, как APT, STRESS, АДАРТ (США), SYMAP (ГДР), САР, АПРОКС (СССР).

В настоящее время имеется тенденция развития специализированных языков с проблемной ориентацией. Это языки более высокого уровня, чем алгоритмические языки с процедурной ориентацией. В программах, составленных на алгоритмических языках с процедурной ориентацией (АЛГОЛ, КОБОЛ), программист должен точно определить используемые в вычислениях процедуры и подробно их описать. В специализированных языках с проблемной ориентацией операторы языка имеют командную структуру (макро), записанные в терминах проблемы и реализующие отдельные процедуры вычислительного процесса. Поэтому программисту не надо дополнительно описывать эти процедуры и следить за порядком их следования. Благодаря высокой степени декларативности этих языков пользователь имеет возможность решать одну и ту же проблему различными методами в зависимости от его опыта в данной специальности, а не в программировании.

Другими примерами таких систем могут служить генераторы отчетов, генераторы сортировок, табличные языки и др. В задачу программиста при работе с генератором отчетов входит задание информации о структуре и местонахождении входного массива информации, формате и структуре требуемого отчета. При этом имеется в виду, что алгоритм получения отчета не описывается.

Существуют две разновидности генераторов отчетов. Первая разновидность предусматривает наличие инвариантных программ, настраивающихся по информации о входных массивах и отчете. После выполнения настройки такой генератор обеспечивает ввод исходного массива и изготовление отчета.

Вторая разновидность — генератор отчетных программ на базе информации о входных массивах и отчета, генерирует рабочую программу, выполнение которой обеспечивает получение отчета заданного вида.

Для любой из разновидностей генераторов отчетов требуется задание следующих групп информации:

описание ввода, включающее описание входных документов и признака, по которому различается тип документа;

описание данных, включающее описание элементов информации, составляющих исходную запись, их выделение для реализации операции, а также описание процессов получения сумм элементов исходных записей;

описание вычислений, включающее описание вычислительных схем для получения показателей отчета на основе исходных данных;

описание вывода, включающее определение формата отчета с учетом заголовка, шапок, нумерации страниц и других особенностей оформления выходного документа.

Естественно, что в подобных системах программирования отладка программ практически исключена, что в значительной мере сокращает время от постановки задачи до ее машинной реализации. Однако необходимым элементом генераторов и других проблемных систем программирования является контроль соблюдения условий задания исходной информации.

Генератор программ отчетов имеет собственный язык РПГ, предназначенный для составления программ, результатом выполнения которых являются разнообразные отчеты начиная от простого перечня или ведомости до полного отчета, содержащего итоговые данные по различным показателям и напечатанного по требуемой форме. Достоинство этого языка в том, что от программиста не требуется знания машин, он должен описать только задачу, не описывая подробного алгоритма ее решения.

Для описания задачи в РПГ существует несколько типов бланков, на которых описываются вводные и выводные файлы и те операции, которые надо произвести над ними. Файлы могут иметь последовательную, прямую или индексно-последовательную организацию. Транслятор РПГ переводит исходную программу в объектный модуль, что позволяет написать основную программу на языке РПГ, а отдельные ее части на других языках.

Генераторы сортировок обеспечивают выбор оптимальных алгоритмов упорядочения информации в соответствии с заданным законом на основании параметров сортируемых файлов (количество сортируемых записей, длина записи, длина признака сортировки).

В структурном плане генератор сортировок включает управляющую программу и набор модулей, реализующих различные алгоритмы сортировки информации. Управляющая программа по заданным параметрам сортируемого файла обеспечивает подключение одного из модулей, а после завершения упорядочения обеспечивается вывод полученного файла на заданный носитель информации (магнитные ленты, перфокарты и т. д.).

Генератор сортировок позволяет сортировать файлы, состоящие из неупорядоченных записей, а также объединять несколько файлов с упорядоченными записями в один упорядоченный последовательно организованный файл. Записи могут быть рассортированы в убывающей или возрастающей последовательности. Признаки, по которым производится упорядочение, называются управляющими полями. По каждому управляющему полю упорядочение может производиться в своей последовательности. В процессе выполнения программы сортировки создаются контрольные точки, что позволяет при необходимости возобновить выполнение программы с фиксированной промежуточной точки.

Параметры и характеристики сортируемых файлов задаются с помощью управляющих операторов сортировки. В управляющих операторах указываются такие сведения, как описание файлов, подлежащих сортировке или слиянию, описание управляющих полей, режим работы программы.

В системе, базирующейся на табличных языках, процесс обработки информации и сама обрабатываемая информация описывается с помощью таблицы принятия решений, отражающей логику решения задачи. Закодированные таблицы вводятся в память машины и транслируются в рабочую программу с ее последующим решением. Таблицы принятия решения независимы от машины, исключают словесное описание задачи, составление блок-схемы и программы. Они являются удобным средством общения различных специалистов, работающих над общей проблемой.

Одно из направлений дальнейшего расширения использования таблиц принятия решения — создание простых и гибких процессоров с табличного языка, связанных с одним из процедурных языков программирования, которые обеспечивают получение рабочих программ. Специфика табличных языков, ограничивающих круг решаемых задач, сдерживает их широкое распространение.

Три перечисленных выше системы программирования имеют в своей основе язык и используются для составления программ. Вспомогательные системы включают набор заранее подготовленных программ, выполняющих ряд вспомогательных функций при обработке данных. Компоненты вспомогательных систем программирования такие, как системы отладки, используются в совокупности с различными компонентами машинных, процедурных и проблемных систем программирования для обслуживания работы ЭВМ.

Отладочные программы обеспечивают проверку и выявление ошибок в рабочих программах. Их использование не требует знаний о рабочей программе, получаемой с помощью транслятора, или инвариантной программе, получаемой автоматической настройкой по исходной информации.

Средства отладки разделяются на две категории: одни могут быть использованы в процессе выполнения рабочих программ, а другие в тех случаях, когда в программе в процессе выполнения произошла ошибка. Средства отладки, используемые в процессе выполнения программы, обеспечивают вывод на печать промежуточной информации, документируют ход выполнения программы и выдают в явном виде результаты решения задачи. Средства отладки, вызываемые в программу при появлении ошибки, предназначены для локализации ошибки, «узнавания» типа ошибки и выполнения тех действий, которые связаны с ликвидацией последствий возникшей ошибки.

Наиболее удобна отладка программы на уровне входного языка. Существуют отладочные программы, предоставляемые операционной системой, и отладочные программы, связанные с соответствующей системой программирования. Последние учитывают специфику входного языка и соответствующего компилятора, поэтому в данной работе они не рассматриваются. Отладочные средства, предоставляемые программным обеспечением, как правило, работают в режиме интерпретации. Программист вставляет в свою программу на входном языке специальные макрокоманды обращения к одной из

отладочных программ. При формировании загрузочного модуля для выполнения эти макрокоманды дают управляющую информацию для вызова требуемой отладочной программы. В момент выполнения программы отладочный интерпретатор выполняется в соответствующих точках программы. Он выдает требуемую информацию и по окончании своего действия возвращает управление отлаживаемой программе. Программа продолжает выполняться обычным путем, пока не встретится следующий оператор отладки. После завершения отладки достаточно, например, заменить операторы отладки пустым оператором, и программа становится «непрерывной». При этом включение и удаление отладочных средств не требует повторной компиляции.

Кроме выполнения своих основных функций отладочные средства позволяют произвести корректировку программ, т. е. без перетрансляции заменить, добавить или удалить команды программы, построить во вспомогательной памяти файлы, которые требуются отлаживаемой программе как вводные.

Наиболее употребительными отладочными средствами являются отладочная выдача и прокрутка. Отладочная выдача производит распечатку регистров и заданных областей. Прокрутка дает детальную распечатку о выполнении всех команд программы или только команд передачи управления.

8.2. Компиляторы

Программа, обеспечивающая перевод входного языка на машинный, называется транслятором.

В зависимости от функционального назначения транслятор может быть либо компилятором, либо ассемблером, либо интерпретатором.

Под компилятором будем понимать программу, обеспечивающую перевод с алгоритмического языка на машинный без одновременного выполнения получаемой программы.

Под ассемблером понимается транслирующая система с языка ассемблера. При этом типы ассемблеров зависят от видов языка ассемблера. Если одному оператору входного языка соответствует одна машинная команда, то такой ассемблер называется абсолютным, а иногда автокодом. Если существуют макрокоманды, которые переводятся в группу машинных команд, то транслирующая система называется макроассемблером.

Если трансляция исходной программы на входном языке совмещается с выполнением данной программы, то такая транслирующая система называется интерпретирующей или интерпретатором. Входным языком интерпретатора может быть ассемблерный язык, проблемно-ориентированный или специализированный язык. На вход интерпретатора поступает последовательность операторов входного языка и он определяет, какую из подпрограмм нужно выбрать и выполнить, если возможно, вместо каждого оператора. Интерпретато-

ры используются для трансляции и выполнения программ с исходных языков, моделирования работы одной вычислительной машины на другой, моделирования различных систем, отладки программ.

Транслирующие системы со специализированных языков высокого уровня многими авторами называются процессорами. Они представляют собой более сложную организацию транслятора и могут одновременно совмещать в себе элементы компилирования, интерпретирования и ассемблирования.

В общем случае любая система программирования обеспечивает следующие функции: контроль правильности записи алгоритмов и программ на входных языках и выдачу информации о наличии, месте и характере ошибок, общее распределение памяти и описание глобальных переменных, используемых многими подпрограммами данной задачи; трансляцию отдельных частей или всего алгоритма, написанного на входном языке, в некий промежуточный язык низкого уровня или машинные коды; автоматическую стыковку подпрограмм внутри отдельно протранслированных частей общего алгоритма по глобальным переменным; накопление в одной из библиотек системы результатов трансляции отдельных частей алгоритма для дальнейшего объединения их в загрузочные модули; выпуск сопровождающей технической документации (распечатки программ на входном и машинных языках, сведения о распределении памяти и другие справочные материалы).

Изучение трансляторов начнем с рассмотрения компиляторов.

В процессе компиляции решаются следующие задачи:

1) распознание конструкций входного языка и в случае необходимости перекодирование их в промежуточный язык;

2) анализ структуры исходной программы с целью определения области действия и состава операторов;

3) обработка символических имен (идентификаторов), обеспечивающая связку описаний величин и их вхождения в операторы;

4) программирование выражений в последовательность простых операций;

5) распределение памяти под программу и данные;

6) редактирование и вывод документации по задаче.

В зависимости от способа получения рабочей программы все компиляторы делятся на одно- и многопроходные.

В общем случае однопроходный компилятор включает блоки синтаксического анализа, блоки анализа структуры данных, генератор рабочей программы. Пример структуры компилятора приведен на рис. 8.1. Сплошные стрелки на этом рисунке соответствуют линиям связи, пунктирные стрелки указывают на доступ к таблицам. Если пунктирная стрелка указывает на таблицу, то это означает, что последняя строится. Если пунктирная стрелка идет от таблицы, то это означает, что таблица используется соответствующим блоком компилятора. Детальное описание принципов построения компиляторов приводится в главе 9.

В многопроходных компиляторах последовательность простых

преобразований применяется по очереди ко всей исходной программе. Многопроходную систему можно рассматривать в общем виде как совокупность различных компиляторов, применяемых к исходной программе (например, компиляция с языка фразовой структуры в язык ассемблера и далее в машинный код).

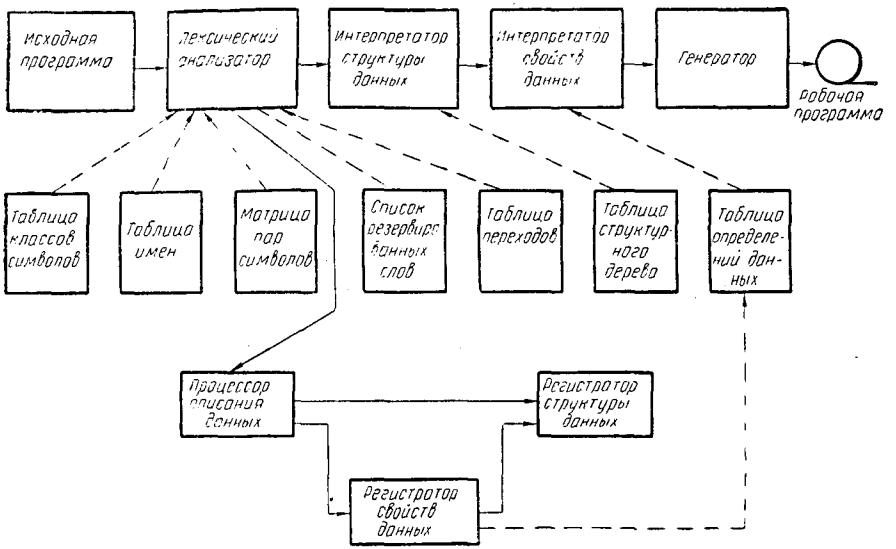


Рис. 8.1. Структура компилятора

В многопроходном компиляторе за счет увеличения числа просмотров собирается значительно больше информации об исходной программе по сравнению с информацией, полученной при однопроходной компиляции. Эта избыточная информация может быть использована для получения более эффективной рабочей программы из одной и той же исходной программы. В более совершенных многопроходных компиляторах применяют дополнительные просмотры всей программы для выделения случаев, в которых возможна оптимизация.

Когда большие потери времени в оптимизирующих компиляторах не оправдываются (например, во время отладки) полезен однопроходный компилятор с простой и быстрой схемой работы.

Возможно дальнейшее разбиение класса многопроходных компиляторов: вынужденно многопроходный и многопроходный оптимизирующий.

Одной из причин разработки многопроходного компилятора может явиться малый объем оперативной памяти машины, не разрешающий разметить в ней весь сложный аппарат схемы компиляции. Тогда весь алгоритм компиляции делится на более или менее рав-

ные части, между которыми распределяется алгоритм. Следствием такого разделения является возможность получения на каждом из просмотров избыточной информации, которая может быть использована для некоторой оптимизации рабочей программы.

Требование эффективности рабочей программы может быть основной причиной многопроходности компилятора, не зависящей от емкости машины. В этом случае важнейшей функцией генерации рабочей программы является дополнительный анализ промежуточных программ на возможность их оптимизации.

Все компиляторы, входящие в состав программного обеспечения вычислительной системы, работают под управлением операционных систем и должны удовлетворять некоторым принятым соглашениям. Программа, написанная на любом входном языке, имеющемся в системе, транслируется в объектный модуль, который представляет собой программный модуль в промежуточном формате, общем для всех компиляторов системы. Специфика исходного языка программирования после трансляции теряется. Объектный модуль не является программой, которую можно выполнить на машине, так как он может содержать символические адреса, неопределенные во время трансляции, а также некоторую другую информацию.

Объектные модули, прошедшие этап редактирования (обработанные одной из управляющих программ редактором), называются загрузочными модулями. Загрузочные модули готовы к выполнению на машине. Они могут состоять из нескольких объектных модулей. Объединение объектных модулей в загрузочные происходит независимо от того, когда и с какого языка протранслирован тот или иной модуль. Это позволяет различные части задачи программировать на наиболее подходящем для нее языке.

8.3. Ассемблеры

Под ассемблером будем понимать программу, обеспечивающую перевод с входного машинно-ориентированного языка на машинный. В общем случае в языке ассемблера существует пять типов команд: 1) мнемонические, 2) псевдокоманды, 3) макрокоманды, 4) квазикоманды, 5) условные команды ассемблера.

Мнемоническая команда соответствует одной машинной команде. В этой команде вместо машинных кодов операций используются мнемонические обозначения. Иногда эти команды называются машинными.

Псевдокоманды служат для передачи информации ассемблеру, а не для вставления их в программу на машинном языке (например, для отделения программы друг от друга, резервирования памяти). Псевдокомандам в программах на машинном языке не соответствует никаких инструкций.

Квазикоманды порождают в рабочей программе информацию, предназначенную для использования другими частями программного обеспечения. Например, во время загрузки программы в память

загрузчику передается информация о работах, которые он должен выполнить. Квазикоманде в машинной программе также не соответствует никаких инструкций.

Условные команды ассемблера используются для управления процессом трансляции. Макрокоманды заменяются на несколько команд машинного языка. Наиболее простыми являются ассемблеры, состоящие из мнемонических команд.

Общий вид мнемонической команды можно представить следующим образом:

Поле имени	Поле операции	Поле операндов	Примечание

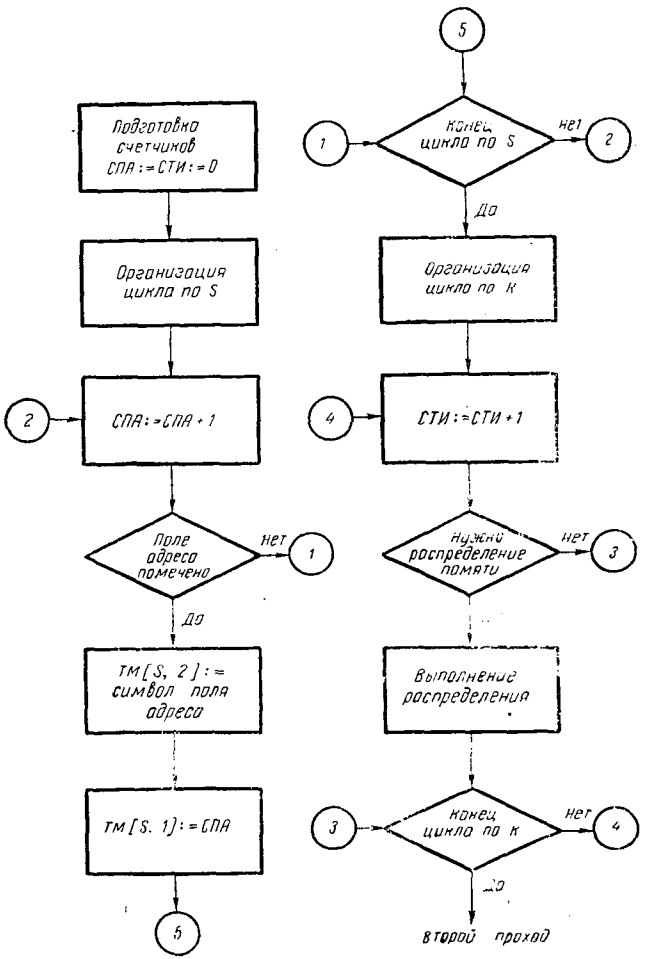


Рис. 8.2. Схема первого прохода ассемблирования

Поле имени может содержать либо имя, либо метку очередной инструкции. Такие метки позволяют ссылаться на команду как на операнд. В поле операции записываются символы, определенные языком ассемблера и соответствующие машинным кодам операций. В поле операндов записываются либо адреса, либо идентификаторы операндов, которые ассемблером переводятся в машинные адреса.

В примечании программист записывает свои замечания. Эти примечания не становятся частью программы, а могут выдаваться на печать при трансляции.

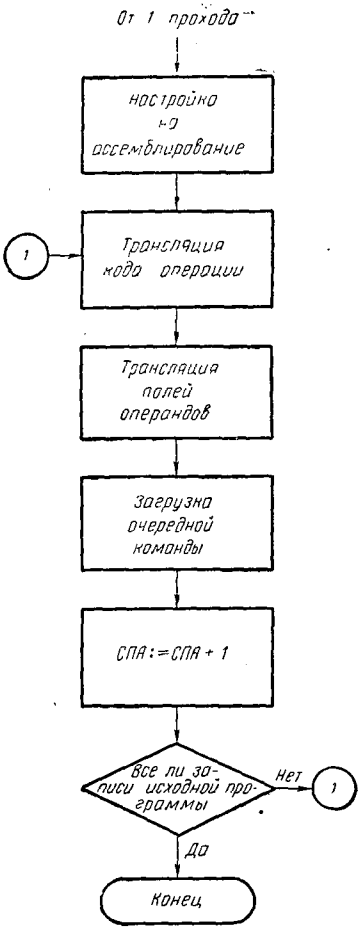


Рис. 8.3. Схема второго прохода ассемблирования

При трансляции исходной программы основная трудность состоит в том, что обычно ассемблер рассматривает команды исходной программы последовательно одну за другой, так как всю программу сразу он охватить не может. Но в некоторой части программы могут появляться ссылки на метки, которые еще не определены и местоположение которых еще не известно. Эта особенность, а также последовательный характер выполнения команд приводят к тому, что большинство ассемблеров выполняют свою работу за два просмотра программы на входном языке, называемых часто проходами. Проход — это отдельный просмотр всей исходной программы.

Цель первого прохода — проверка правильности исходной программы, составление таблиц, распределение памяти, распечатка обнаруженных ошибок. Принципиальная схема первого прохода приведена на рис. 8.2. В схеме приняты следующие обозначения:

СПА — счетчик полей адресов,

ТМ — таблица меток,

СТИ — счетчик таблиц имен,

К — текущая координата таблицы имен.

Во втором проходе выполняется трансляция мнемонических обозначений и размещение данных, для которых в первом проходе было проведено распределение памяти (рис. 8.3).

Таким образом, в ассемблере с двупроходной трансляцией при первом проходе исходной программы заполняется ряд таблиц. Например, в ассемблере, содержащем только мнемонические команды, на первом просмотре выполняется

построение таблицы помеченных полей, структура которой приведена ниже.

Имя	Состояние счетчика адреса
...	...

Общая блок-схема трансляции будет значительно сложнее, если в языке ассемблера будут иметь место любые другие разновидности команд.

В зависимости от сложности языка могут составляться следующие таблицы:

таблица команд для перевода мнемонических обозначений в коды команд;

таблица ошибок для фиксации обнаруженных ошибок (иногда вместо формирования такой таблицы осуществляется распечатка);

таблица эквивалентных имен для организации перекрестных ссылок эквивалентных имен;

таблица распределения памяти.

В однопроходных ассемблерах последовательность простых преобразований применяется по очереди к каждой небольшой части таким образом, что машинная программа образуется в процессе единственного просмотра программы. Однако создание однопроходных ассемблеров накладывает ограничения либо на размер исходной программы (она целиком должна размещаться в оперативной памяти), либо на язык ассемблера (он должен быть элементарно простым).

Наиболее эффективные машинные программы создаются с использованием многопроходных ассемблеров.

8.4. Интерпретаторы

Общий процесс интерпретации заключается в чередовании программирования и счета отдельных частей задачи на ЭВМ. Режим интерпретации предполагает наличие исходной программы, написанной на специальном языке, и набора модулей, осуществляющих трансляцию отдельных конструкций языка и их выполнение. Таким образом, интерпретаторы читают, расшифровывают и выполняют операторы языка.

Известны следующие случаи использования интерпретаторов:

моделирование реального вычислителя, т. е. моделирование на реальной машине операций существующей или разрабатываемой ЭВМ;

моделирование гипотетической машины с целью изучения приемов ее использования, например симуляция обработки списков на «несимвольной» машине;

интерпретация языков моделирования, предназначенных для описания параллельных процессов, на последовательных машинах;

интерпретация последовательности управляющих инструкций в целях управления групповыми процессами обработки, процессами реального времени и разделения времени;

интерпретирующие трансляторы, обеспечивающие выполнение программ на исходных языках непосредственно через интерпретатор.

Интерпретаторами часто пользуются, когда проектируется новая машина и на нее создается новая система программирования.

Практика проектирования интерпретаторов показывает, что их трудно писать, отлаживать и модифицировать. Кроме того, интерпретаторы обладают малой скоростью работы, требуют постоянного присутствия в поле оперативной памяти вместе с интерпретируемой программой и сравнительно велики по объему.

По этим причинам интерпретаторы пишутся: для научно-исследовательских целей; для языков, не имеющих четко выраженной структуры; для очень мощных систем; для моделирования различных процессов.

На практике более целесообразно строить трансляторы не в режиме интерпретации, а в сочетании с методами компиляции и ассемблирования. Примером такого сочетания может служить процессор со специализированного языка высокого уровня.

ВОПРОСЫ К ГЛАВЕ

1. Классификация систем программирования.
2. Машинные системы программирования и их назначение.
3. Проблемные системы программирования и их краткая характеристика.
4. Процедурные системы программирования и их классификация.
5. Вспомогательные системы программирования и их особенности.
6. Определение транслятора, компилятора, ассемблера, интерпретатора, процессора.
7. Однопроходные компиляторы, их преимущества и недостатки.
8. Многопроходные компиляторы, их особенности.
9. Тип команд, используемых в ассемблерах, и их краткая характеристика.
10. Интерпретаторы и их использование.

Глава 9

ТРАНСЛЯЦИЯ

9.1. Введение в трансляцию

Транслятор может быть рассмотрен как устройство (прибор), которое трансформирует входную строку A в выходную строку B в соответствии с некоторой функцией преобразования T:

$$B := T(A).$$

Независимо от того, будет ли выходная строка B выдаваться в машинном коде или нет, задача транслятора не определена до тех пор, пока не будет принято решение, каким образом представлять различные элементы входной строки A, написанной на алгоритмическом языке, в рабочей программе B. Вследствие этого границы между преобразованием в рабочую программу и ее выполнением несколько стираются и могут значительно отличаться друг от друга в различных трансляторах.

Однако функцию T — преобразования исходной программы, записанной на алгоритмическом языке, в форму, пригодную для последующего использования на машине, можно определить по следующим этапам:

1) редактирование исходной программы с целью распознания различных элементов входного языка;

2) анализ структуры исходной программы;

3) установление взаимосвязи между используемыми идентификаторами и их описаниями;

4) преобразование различных выражений входного языка с учетом правил старшинства и расположения скобок в некоторую последовательность простых операций.

Этапы 1 и 4 — общие для трансляторов с любых алгоритмических языков. Алгоритмы редактирования этапа 1 существенно зависят от фактического способа представления исходной программы, что, в свою очередь, определяется набором устройств подготовки данных. Этапы 2 и 3 характерны для языков, имеющих блочную структуру (АЛГОЛ, АЛГЭК и другие алголоиды).

В соответствии с этими этапами транслятор можно разделить на несколько функционально самостоятельных частей, каждая из которых выполняет конкретную работу.

Одной из таких частей транслятора является блок лексического анализа, решающий задачи редактирования исходной программы, включающий опознание элементарных конструкций алгоритмического языка (идентификаторы, числа, ограничители и другие единицы языка) и перекодирование этих конструкций в промежуточный язык, приспособленный для последующей работы.

Непосредственно к лексическому анализу примыкает задача анализа исходной программы в целях определения области действия и состава операторов.

На этапе синтаксического анализа осуществляется контроль синтаксической корректности программы.

Программирование выражений в последовательность простых операций осуществляется специальным блоком генерирования машинных команд. С учетом предыдущих действий с операторами исходной программы для большинства из них процесс генерации будет простым. Часть операторов (например, операторы обращения к процедуре) заменяется стандартными шаблонами с небольшим числом переменных полей, настраиваемых в зависимости от текущих значений переменных. В процессе генерирования рабочей програм-

мы обычно включаются в работу схемы оптимизации в зависимости от целевой функции эффективности (скорость счета по рабочим программам, расход оперативной памяти и другие критерии).

Вслед за генерированием машинной программы выполняется распределение памяти для ее работы. Задача распределения памяти решается в случае, если выходным языком транслятора является язык конкретной ЭВМ. Основная проблема распределения памяти — сегментация рабочей программы и исходных данных в связи с ограниченным объемом оперативной памяти машины. Задача редактирования рабочей программы и ее последующего документирования включает перекодирование символических имен в действительные с перекомпоновкой программы и ее последующим выводом на заказанный набор внешних устройств.

В зависимости от способа получения рабочей программы все трансляторы делятся на одно- и многопроходные системы.

Главная проблема, возникающая при разработке трансляторов для экономических задач, — значительное разнообразие и противоречивость задач, решение которых требуется. Рассмотрим основные из них.

1. *Легкость обучения.* Документация системы должна включать элементарное руководство по программированию и средства, необходимые для программирования реальных и сложных процессов переработки экономической информации.

При этом от компилятора требуется выявление ошибок пользователя, допущенных последним в процессе написания программ. Это требование может быть выполнено за счет снижения эффективности трансляции (увеличение затрат времени, ограничение возможностей пользователя) путем разработки алгоритмов тщательной проверки исходных и рабочих программ и выдачи четких сообщений об ошибках с рекомендациями для исправлений.

2. *Отделение описания данных от процедур преобразования.* Разработчик программы должен сделать описание структур данных отдельно от процедур их преобразования, так как данные могут существовать независимо от программы. Это требование означает, что необходимо хранить управляющий массив описаний, доступный многим программам пользователей, и в это же время обеспечивать средства ведения этого массива (расширение, корректировка), а также возможность выдачи сообщений о его состоянии. Это требование особенно остро стоит в задаче автоматизированных систем управления при создании банков данных АСУ.

3. *Определение иерархических структур входной и выходной информации.* Сложная иерархическая структура входных и выходных данных экономических задач требует от разработчиков компиляторов решения трудных вопросов размещения массивов данных и их вызова на обработку.

Наличие в языках описания обработки данных широкого диапазона форматов и представлений обеспечивает пользователям возможность работы теми же средствами и с теми же условными обо-

значениями, которыми они обычно пользуются. Эти обозначения, как правило, не приспособлены для машинной обработки, что создает дополнительные трудности для авторов компиляторов.

4. *Эффективность.* Экономические задачи обладают более устойчивой периодичностью решения, чем задачи научно-технические. Среди типовых задач АСУ можно встретить представительный набор задач с ограниченным сроком обработки информации. Поэтому от трансляторов с языков описания обработки экономической информации требуется получение более эффективных программ с точки зрения расхода машинного времени.

5. *Выявление характеристик транслятора и рабочей программы.* К характеристикам транслятора относят описание деталей использования транслятора, ввода исходных программ с различных внешних устройств, составление информации о характере действий при обнаружении ошибок, а также способы документирования, предусмотренные в трансляторе.

К характеристикам рабочей программы относится информация об ошибках с указанием ссылки на их место в программе.

9.2. Лексический анализ при трансляции

Сложность алгоритмов лексического анализа определяется набором изобразительных средств исходного языка и наличием соответствия между этими средствами и возможностями устройств подготовки данных по перенесению текста исходной строки на носители для ввода в память. При лексическом анализе прежде всего решается задача перекодирования из символики исходного языка на язык промежуточных символов. Задача перекодирования при условии однозначного соответствия символов входного и промежуточного языков решается простым поиском эквивалентов с последующей заменой.

Входная информация для трансляции может вводиться посимвольно, построчно или как единый массив. Разбор введенного текста обычно выполняется посимвольно.

В случае посимвольного ввода программы на входном языке программа лексического анализа (лексический анализатор) должна обеспечить опознание принадлежности текущего символа к одной из выделенных групп: идентификаторы, числа, строки, подчеркнутые символы входного языка, зарезервированные слова, ограничители и т. д. Перекодирование в этом случае включает этап выбора таблицы, по которой оно будет выполняться, и поиска в этой таблице.

Таблицы перекодирования обычно строятся из записей, состоящих из трех частей. В первой части записи размещается конструкция входного языка. Вторая часть содержит эквивалент промежуточного языка и в третьей части содержится отсылка на вспомогательную программу, которая должна быть выполнена после перекодирования символа. В качестве такой программы может использо-

ваться, например, программа вызова очередного символа исходной строки на обработку.

В процессе лексического разбора текста исходной строки может быть обнаружена ошибка в подготовке информации. При обнаружении ошибки необходимо сформировать и выдать на печать информацию о характере ошибки, ее местоположении и обеспечить продолжение перекодирования со следующего основного символа или ограничителя. Важно за один выход на трансляцию выявить большую часть ошибок в исходной программе. При обнаружении ошибок подготовки информации работа по трансляции продолжается передачей управления блоку синтаксического контроля и всем последующим блокам транслятора. Вероятность того, что в результате синтаксического контроля не будут выявлены ранее зафиксированные ошибки подготовки, практически мала, так как существующие алгоритмы синтаксического контроля работают с высокой надежностью.

Кроме распознания и перекодирования символов на лексическом уровне, организуется построение таблицы идентификаторов (ТИ), встречающихся в исходной программе. При этом реализуется алгоритм замены многосимвольных конструкций идентификаторов (часто количество символов, по которым различаются идентификаторы, оговаривается) адресом их местоположения в ТИ, т. е. происходит резкое сокращение объема исходной программы и стандартизация длины для всех ее объектов. ТИ представляет собой секционную таблицу, каждая из секций которой содержит однотипную информацию, например секция массивов типа **real** и т. д.

На этапе лексического анализа создаются также таблицы констант различных типов, меток, процедур переключателей. При засылке констант в соответствующие таблицы организуется их перекодирование в принятую в данном трансляторе систему счисления и контроль на соблюдение допустимых пределов величин констант. Выходом лексического анализатора является преобразованная исходная программа на промежуточном языке.

9.3. Синтаксический контроль

Основной целью синтаксического анализа является проверка правильности промежуточной программы, полученной лексическим анализатором. Для выполнения этой работы используются различные алгоритмы грамматического разбора, получившие в последние годы теоретическое и практическое развитие.

Среди множества алгоритмов грамматического разбора с точки зрения практической реализации представляет интерес алгоритм грамматического разбора для грамматик с операторным предшествованием.

Отношения предшествования между терминальными символами изображаются таблицами предшествования, на базе которых осуществляется контроль синтаксиса путем последовательной свертки

понятий. Запись в таблице предшествования может содержать один из знаков $< \cdot$, $\cdot >$, $\doteq$ или быть пустой. Пустая запись означает, что в рассматриваемом языке имеются два оператора, которые не могут быть расположены рядом. Если очередная пара терминальных символов связана согласно таблице предшествования отношением предшествования $< \cdot$, то производится прием очередных символов из исходной строки программы. Если очередная пара символов связана отношением предшествования $\cdot >$, то выполняется замена в стеке связки двух символов, соединенных знаков операции, на их ожидаемый результат. Так выполняется последовательная свертка понятий до получения, например, по арифметическому оператору только одного символа — значения результатов. Фрагмент таблицы предшествования применительно к арифметическим операциям «+» и «×» приведен в табл. 9.1.

Таблица 9.1

	+	×	(	)	⊣
⊢	$<\cdot$	$<\cdot$	$<\cdot$		
+	$\cdot>$	$<\cdot$	$<\cdot$	$\cdot>$	$\cdot>$
×	$\cdot>$	$\cdot>$	$<\cdot$	$\cdot>$	$\cdot>$
(	$<\cdot$	$<\cdot$	$<\cdot$	$\doteq$	
)	$\cdot>$	$\cdot>$		$\cdot>$	$\cdot>$

Если в процессе просмотра текущей пары символов по таблице предшествования будет установлено отсутствие операции отношения, то этот факт (в упрощенном виде) свидетельствует о наличии синтаксической ошибки. Факт наличия ошибки фиксируется, формируется сообщение оператору об обнаруженной ошибке и производится пропуск символов исходной строки до такого символа, с которого можно снова организовать контроль. Таким символом, например, в арифметическом выражении может быть символ точка с запятой в конце выражения.

Так как величины в таблице предшествования могут принимать одно из четырех значений, то для их кодирования достаточно двух двоичных разрядов памяти, поэтому объем памяти, занимаемый таблицей предшествования, невелик. На практике подобная организация имеет ряд недостатков, например строгая определенность действий, выполняемых при обработке всех записей таблицы, и фиксированный порядок обработки комбинаций исключают введение специальных операций над содержимым стека. Необходимость же таких действий при трансляции диктуется структурой алгоритмических языков.

Одним из путей устранения этого недостатка является такое кодирование значений в таблице предшествования, при котором вместо значения операции предшествования ставится указатель отсылки на специальную подпрограмму, обеспечивающую обработку текущей комбинации символов. Такой подход позволяет выполнять

и нестандартные действия, но объем памяти, занимаемый таблицей предшествования, при этом резко увеличивается.

Среди других алгоритмов реализации синтаксического контроля исходной программы в литературе рассматриваются правила подстановки Флойда, направленный анализ «сверху вниз» и «снизу вверх» и ряд других алгоритмов. Большинство этих методов изучается в курсе «Теория алгоритмов и грамматик».

Общими проблемами в любом из методов являются поиск компактных и быстрых алгоритмов анализа при минимальном объеме диагностической информации, создание словарей типовых ошибочных ситуаций с целью сообщения пользователю не только информации об ошибке, но и ее локализации с предложениями для устранения. Кроме того, весьма важной и нерешенной остается проблема минимизации неконтролируемого участка исходной программы между точкой ошибки и ближайшей точкой продолжения контроля.

9.4. Генерирование рабочих программ

При трансляции с алгоритмических языков одной из основных задач является получение эффективных машинных программ для различных операторов (особенно арифметических) за приемлемое время трансляции. Экономия времени трансляции рассматривается при условии, что транслятором получается качественная рабочая программа, не уступающая по своим характеристикам продукции среднего программиста, разрабатывающего рабочую программу вручную.

Наиболее полно вопросы генерирования рабочих программ разработаны для алгоритмического языка АЛГОЛ-60. Рассмотрим применительно к этому языку некоторые способы генерирования рабочих программ.

Трансляция выражений

В практике трансляции разработано значительное количество методов перевода арифметических выражений из исходной языковой записи к виду, обеспечивающему выполнение вычислений на машине. Рассмотрим некоторые из них.

Методы однократного просмотра

В отличие от ранних методов трансляции методы однократного просмотра используют табличную технику генерации кодов и являются сравнительно быстрыми методами.

Один из таких методов, названный методом Кеннера, обеспечивает в процессе последовательного просмотра исходной строки слева направо занесение в некоторый список как идентификаторов, так и знаков операций. Перед записью очередного знака операции производится анализ списка на возможность прерывания процесса за-

писи и генерацию кода. Прерывание может быть вызвано появлением правой скобки, знака операции или завершением выражения. Кроме того, в этом методе расширены возможности экономного использования промежуточных ячеек памяти.

Так, выражение

$$(A + B) - (C - D) \uparrow E$$

будет переведено в следующую последовательность кодов:

$$R1 := A + B;$$
$$R2 := C - D;$$
$$R2 := R2 \uparrow E;$$
$$R1 := R1 - R2.$$

Среди других методов однократного просмотра более универсальным является метод алгоритмических матриц. Он отличается от ранее рассмотренного направлением просмотра выражения (справа налево) и использованием для переупорядочения арифметических операций в соответствии с правилами старшинства специального вектора очередности старшинства.

В результате просмотра формируется алгоритмическая матрица, каждая строка которой фактически закодирована трехадресной командой, предполагающей выполнение операций над двумя операндами с размещением результата в промежуточной памяти. Алгоритм экономии памяти обеспечивает экономное расходование промежуточной памяти. Например, для предыдущего выражения будет построена следующая алгоритмическая матрица.

Номер строки	Номер столбца			
	1	2	3	4
1	—	C	D	$R1$
2	$\uparrow$	$R1$	E	$R1$
3	+	A	B	$R2$
4	—	$R2$	$R1$	$R1$

Перевод в машинные команды осуществляется с помощью набора матриц перевода по одной на каждую операцию. Одним из достоинств данного метода является его независимость от конкретной машины, так как генерирование машинных программ можно проводить по сменным матрицам перевода.

Метод двойного просмотра

Метод двойного просмотра обеспечивает перевод выражений в эффективные команды одноадресной ЭВМ с индексными регистрами.

Просмотр начинается с левого конца выражения и продолжается до тех пор, пока не будет обнаружена закрывающая скобка для одного или более индексов. После этого направление просмотра изменяется и переводятся индексные выражения. Дальнейший просмотр продолжается слева направо с предыдущей точки прерывания

(закрывающей индексной скобки). Процесс продолжается до завершения просмотра выражения. Окончательный просмотр выражения справа налево обеспечивает построение команд машинной программы.

Таким образом, перевод выполняется справа налево с предварительной заменой индексных выражений.

Пример:

$$(A[i+j\times k]+B)\times(C-D[i-j]).$$

При просмотре слева направо с возвратом получим:

$$R1:=j\times k;$$
$$R1:=i+R1;$$
$$R2:=i-j.$$

После окончательного просмотра справа налево получим:

$$R2:=C-D[R2];$$
$$R1:=A[R1]+B;$$
$$R1:=R1\times R2.$$

Машинная память для данного способа организуется в виде трех векторов; два вектора собирают и хранят информацию об идентификаторах, получаемую при просмотрах слева направо с возвратом. Третий вектор хранит информацию о знаках операций.

Интегральные методы

Ранее рассмотренные методы трансляции арифметических выражений использовались в трансляторах, состоящих из ряда закрытых подпрограмм для обработки выражений и операторов разного типа. Интегральные методы используются в трансляторах, работающих без закрытых подпрограмм, т. е. трансляция выражений осуществляется по общей методике. Общая методика предусматривает использование магазинной памяти для приема, хранения и выдачи операндов и знаков операции в процессе просмотра исходной программы и программирования. Рассмотрим использование магазинной памяти при трансляции арифметических выражений.

Простой метод трансляции при просмотре выражения слева направо для случая работы всех операций с промежуточными результатами из выражения $A+(B\times C-D)$ переводится в следующую последовательность команд:

$$R_1:=A;$$
$$R_2:=B;$$
$$R_3:=C;$$
$$R_2:=R_2\times R_3;$$
$$R_3:=D;$$

$$R_2 := R_2 - R_3;$$
$$R_1 := R_1 + R_2.$$

Эти команды распадаются на два типа: команды, заполняющие ячейки промежуточных результатов, и команды, выполняющие операции над содержимым двух последовательных ячеек промежуточных результатов. Приведенную выше последовательность команд можно сократить, исключив ссылки к ячейкам промежуточных результатов и переписав в последовательность A, B, C, $\times$, D, —, +. Последовательность может быть использована в качестве команд, определяющих порядок операций с магазинной памятью в процессе вычислений, т. е. любой операнд, считываемый из последовательности, приводит к выполнению операций:

$$R_p := <\text{операнд}>;\ P := P + 1;$$

в то время как знак операции ($*$) вызывает действия:

$$P := P - 1;\ R_{p-1} := R_{p-1} * R_p.$$

Для группы ячеек промежуточных результатов используется термин «магазин» (стек), а для индекса P — «указатель магазина» (указатель стека).

Последовательность символов, используемая для управления стеком, получается путем переписывания исходного выражения в Польскую Инверсную Запись (ПОЛИЗ), т. е. в бесскобочную запись, когда любой операции предшествуют операнды, над которыми она выполняется. Эта запись введена польским логиком Лукашевичем. Таким образом, выражение $A+B$ в ПОЛИЗ запишется: A, B, +, а выражение $(A+B)\times C$ запишется как A, B, +, C, $\times$.

ПОЛИЗ является основой интегральных методов трансляции выражений.

Последние разработки транслирующих систем используют, как правило, стековую технику для компиляции выражений.

Трансляция простых арифметических выражений

Ранее было показано, что из исходной языковой программы легко строить программу на языке ассемблирования (или языке машины) в случае, если выражения преобразованы в ПОЛИЗ.

Сущность преобразования выражений в ПОЛИЗ состоит в том, что ограничители переупорядочиваются относительно переменных и констант с учетом старшинства операций. Например, выражение $A+B\times C\div D$ преобразуется в A, B, C, $\times$, D, $\div$, +.

Для переупорядочения ограничителей может быть использована магазинная память (стек), служащая для временного приема, хранения и выдачи ограничителей. Прием ограничителей в стек и выдача в выходную программу производятся в соответствии с приоритетом последних, т. е. с появлением очередного ограничителя из исходной программы с приоритетом, не меньшим, чем приоритеты

ограничителей в стеке, последние выводятся в ПОЛИЗ и только после этого ограничитель из исходной программы помещается в ПОЛИЗ. Простой аналогией работы стека является сортировочный узел железнодорожной станции.

Сортировочный узел имеет форму Т-образного разъезда с веткой (стеком) для выполнения маневрирования или переупорядочения (рис. 9.1).

Идентификаторы или константы проходят от входа прямо к выходу по пути *i*. Ограничители попадают от точки *M* (стрелки) на выход только через стек по пути *j*. Прежде чем очередной ограничитель будет принят в стек, разрешается сокращение стека за счет ограничителей, имеющих приоритет равный или больший приоритета очередного ограничителя. Эти ограничители по пути *p* попадают в ПОЛИЗ.

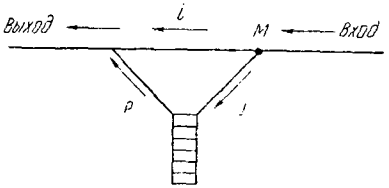


Рис. 9.1. Модель стека

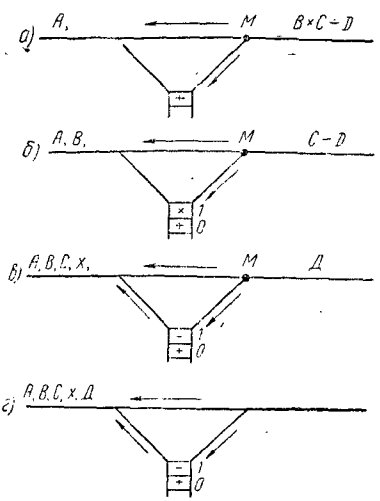


Рис. 9.2. Механизм работы стека

Так, для предыдущего примера при значениях приоритета ограничителей:

+	−		0
×	/	÷	1
↑			2

Состояние схемы после прохождения ограничителем «+» точки *M* представлено на рис. 9.2а.

Идентификатор *A* проходит непосредственно на выход. Ограничитель «+» не вызывает действий по выталкиванию из стека, так как последний пуст. К моменту прохода ограничителем «×» точки *M* состояние схемы имеет вид, представленный на рис. 9.2б. Так как ограничитель «×» имеет приоритет, больший, чем приоритет +, то сокращения стека не происходит. Очередной идентификатор *C* проходит на выход. С поступлением в точку *M* очередного ограничителя «÷», имеющего приоритет, равный приоритету «×», вызывается сокращение стека и выдача на выход ограничителя «×». После этого ограничитель «÷» разместится в стеке. Состояние схемы представлено на рис. 9.2в.

Ограничитель D проходит к выходу, и схема устанавливается в состояние, представленное на рис. 9.2г.

Так как исходное выражение закончено, т. е. на входе нет элементов, то стек очищается. Преобразованное выражение имеет вид: A, B, C, $\times$, D, $\div$, $+$.

В конкретных транслирующих системах используется не один, а несколько стеков (например, стеки величин, типов величин и операций). В этом случае работа транслирующей системы строится следующим образом. Величины от точки M проходят в стек величин, а ограничители — в стек операции. Если очередной ограничитель вызывает сокращение стека на одну позицию, то стек величин сокращается на две последних позиции, и производится программирование очередной ассемблерной команды.

В случае одноместной операции знак «+» игнорируется, а знак «—» заменяется на операцию ИЗ, которой необходимо присвоить приоритет 1, т. е.

+	—			0
×	/	÷	ИЗ	1
↑				2

В связи с динамикой записи арифметических выражений в скобочной структуре необходимо учитывать приоритеты скобок (и). Ограничитель «(» должен попадать в магазин, не вызывая его сокращения. В это же время, ограничителю «)» необходимо приписать приоритет, чтобы с его приходом очистить стек вплоть до ограничителя «(», не выдавая последний в ПОЛИЗ. Сам ограничитель «)» в стек не записывается.

Для завершения передачи содержимого стека после окончания очередного арифметического выражения необходимо иметь специальный символ типа закрывающей скобки, с появлением которого должна включаться процедура очистки стека. Таким символом является «;» с приоритетом 1.

На основании вышеизложенного ряд приоритетов теперь должен быть таким (см. табл.).

Ограничитель	Приоритет
(	0
+ —) ;	1
× / ÷ ИЗ	2
↓	3

Трансляция простых логических выражений

Согласно синтаксису логического выражения и старшинству операции для выполнения трансляции необходимо внести логические операции отношения в таблицу приоритетов.

Любая операция отношения должна иметь приоритет меньше приоритета арифметических операций, т. е. при появлении в стеке символа операции отношения необходимо «вытолкнуть» из стека

все ограничители, относящиеся к арифметическому выражению, предшествующему операции отношения.

Символы логических операций при появлении в стеке должны «выталкивать» ограничители операций отношения, т. е. знаки логических операций должны иметь меньший приоритет по сравнению с операциями отношения.

С другой стороны, логическое выражение может быть заключено в круглые скобки, а, следовательно, приоритет символов (и) должен оставаться соответственно 0 и 1.

С учетом сказанного выше таблица приоритетов имеет вид:

Ограничитель	Приоритет
(	0
,	1
⊃	2
∨	3
∧	4
¬	5
> ≥ < ≤ = ≠	6
+ −	7
× / ÷ ИЗ	8
↑	9

Для обработки переменных с индексами необходимо включить в таблицу приоритетов символы «[», «]» и «,».

Индексным скобкам задают приоритет как для соответствующих круглых скобок. Ограничитель «,» должен выполнять, с одной стороны, роль закрывающей скобки для предшествующего ему выражения, а с другой, — роль открывающей скобки для следующего за ним выражения. В роли закрывающей скобки этот ограничитель должен иметь приоритет 1 и обеспечивать выталкивание из стека ограничителей с большим приоритетом, при этом сам ограничитель не попадает в стек. В роли открывающей скобки ограничитель «,» должен попасть в стек и не производить выталкивания. Его приоритет в этом случае 0. Обычно ограничитель «,» в стек не записывается, а является «особым» символом, управляющим работой стека в точке *M*.

Трансляция различных операторов языка

Условные операторы

Для обеспечения трансляции условных операторов с использованием стека необходимо включить ограничители **if, then, else** в таблицу приоритетов.

Ограничитель **if** действует как открывающая скобка для логических выражений, следующих за ним (приоритет его 0).

Закрывающей скобкой для **if** является ограничитель **then**, он одновременно является и открывающей скобкой для последующего выражения. Логическое выражение, расположенное между ограничителями **if** и **then**, в качестве своего значения будет иметь **true** или **false**. Если это **true**, то управление передается оператору, следую-

щему за **then**, а оператор за **else** должен пропускаться (обходиться). Если это значение **false**, то выражение за **then** пропускается, а управление получит оператор за **else**. Для обхода ветвей после **then** и **else** возникает необходимость в формировании команд условного перехода (УП) и ввод служебных меток, а также команды безусловного перехода (БП) перед оператором или группой операторов, стоящих после **else**. Таблица приоритетов имеет вид:

Ограничитель	Приоритет	Сравнительный приоритет
([	0	не имеет значения
if	0	не имеет значения
then	0	1
else	1	2
) , ;]	не имеет значения	1
≡	3	3
⊃	4	4
∨	5	5
∧	6	6
¬	7	7
> ⩾ < ⩽ = ≠	8	8
+ −	9	9
× / ÷ ИЗ	10	10
↑	11	11

Операторы присваивания

Значение выражения справа от ограничителя «: = » присваивается простой переменной или переменной с индексами, указанной в левой части оператора.

В таблицу приоритетов ограничитель «: = » заносится с приоритетом 2, так как он должен выталкиваться ограничителем конца оператора «;».

Пример:

$$A\,[i,\ j] : = 0;$$

В момент появления ограничителя «;» стек очищается и ПОЛИЗ имеет вид:

$$A,\ i,\ j,\ ],\ 0,\ : =$$

В АЛГОЛе допускается использование операторов многократного присваивания:

$$A : = B : = 0;$$

До тех пор, пока не появится ограничитель за идентификатором B, неизвестно, находится ли B в списке левой части (необходим его адрес) или в правой части (необходимо его значение).

ПОЛИЗ приведенного выше примера будет иметь вид:

$$A,\ B,\ 0,\ :=\ ,\ :=$$

Операторы перехода

<оператор перехода> : : = **go to** <именующее выражение>

При преобразовании в ПОЛИЗ ограничитель **go to** располагается вслед за именующим выражением. Приоритет ограничителя **go to** приравнивается к двум для того, чтобы он оставался в магазине до тех пор, пока не будет закончено преобразование (программирование) всего именующего выражения, например:

go to if A **then** M **else** $M1$

преобразуется в ПОЛИЗ

A, *УПЛ*, $M1$, M, *БПШ*, $M1$, **go to**.

Операторы цикла

Исходная программа, содержащая циклы, преобразуется на этапе редактирования с тем, чтобы последние заменить условными операторами. Преобразование операторов цикла состоит в следующем. Тело цикла выделяется с помощью дополнительных меток (служебные метки). Заголовок цикла в зависимости от его вида заменяется.

Заголовок цикла

for $I := P1, P2 \ldots, PN$ **do**

заменяется на

begin $Q1$; $Q2$; ...; QN; **go to** $M3$ **end**;

Здесь PK—K-й элемент списка цикла, а QK — ряд операторов, соответствующих K-му элементу списка цикла. $M3$ — метка преемника. Вид QK зависит от вида элемента списка цикла PK.

Для элемента списка цикла типа арифметической прогрессии A **step** B **until** C
ряд операторов QK имеет вид:

$L2$: $I := A$;
$L1$: **if** $(B) \times ((C) - 1) \geqslant 0$ **then**
begin go to $M1$, $M2$; $I := I + (B)$; **go to** $L1$ **end**;

Метка $M1$ определяет начало тела цикла и ставится сразу после заголовка цикла. Метка $M2$ является концом тела цикла (меткой обратной связи).

Элементу списка цикла типа арифметического выражения AK соответствует ряд операторов:

$L2$: $I := AK$; **go to** $M1$, $M2$;

Элемент списка типа пересчета *A* **while** *B* заменяется на

L2 : *I* : = *A*; **if** *B* **then begin go to** *M*1, *M*2,
go to *L2* **end**;

Здесь *M*1, *M*2 — дополнительные метки, свои для каждого элемента списка цикла.

Составные операторы

Операторы, заключенные в операторные скобки **begin end**, рассматриваются при преобразовании в ПОЛИЗ и последующем программировании как отдельный оператор.

Операторные скобки обрабатываются, как и круглые скобки, и используются для упорядочения выражений в ПОЛИЗ. Скобка **begin** получает приоритет 1. Действия по **end** эквивалентны в этом случае действиям по ограничителю «;».

9.5. Распределение памяти

Память под величины, принимающие числовые или логические значения, распределяется как во время трансляции, так и во время работы готовой программы. Память, которая распределяется во время трансляции, будем называть в дальнейшем фиксированной. Память, которая распределяется во время работы готовой программы, будем называть динамической.

Каждому блоку соответствует в фиксированной памяти свой участок. Начало участка самого первого блока программы совпадает с началом фиксированной памяти. Если блок является внутренним по отношению к другому (объемлющему) блоку, то его участок находится непосредственно за участком объемлющего блока. Если есть два или несколько блоков, расположенных последовательно (невложенных), то их участки имеют одно и то же начало. Это приводит к тому, что общий объем памяти для системы блоков меньше, чем сумма объемов всех блоков системы, так как одни и те же ячейки используются невложенными блоками.

Участок памяти блока состоит из двух частей. Каждому идентификатору, описанному в данном блоке как простая переменная данного типа или массив переменных с индексами, или как процедура, определяющая значение функции, соответствует своя ячейка в первой части участка данного блока. Соответствие между так описанными идентификаторами и последовательно выбираемыми ячейками участка определяется порядком следования описаний этих идентификаторов в описаниях блока.

Вторую часть участка памяти блока составляют расположенные последовательно участки памяти для величин, локализованных в описанных в данном блоке процедурах. Каждому формальному параметру данной процедуры, перечисленному в списке значений и специфицированному как простая переменная или идентификатор

массива, соответствует своя ячейка участка памяти данной процедуры, начиная с первой ячейки участка. За этими ячейками располагаются участки блоков, входящих в тело процедуры.

Таким образом, если в данном блоке описано n простых переменных, m массивов, l процедур-функций, k процедур (включая и процедуры-функции), то длина участка памяти, которая потребуется данному блоку в фиксированной памяти, будет равна

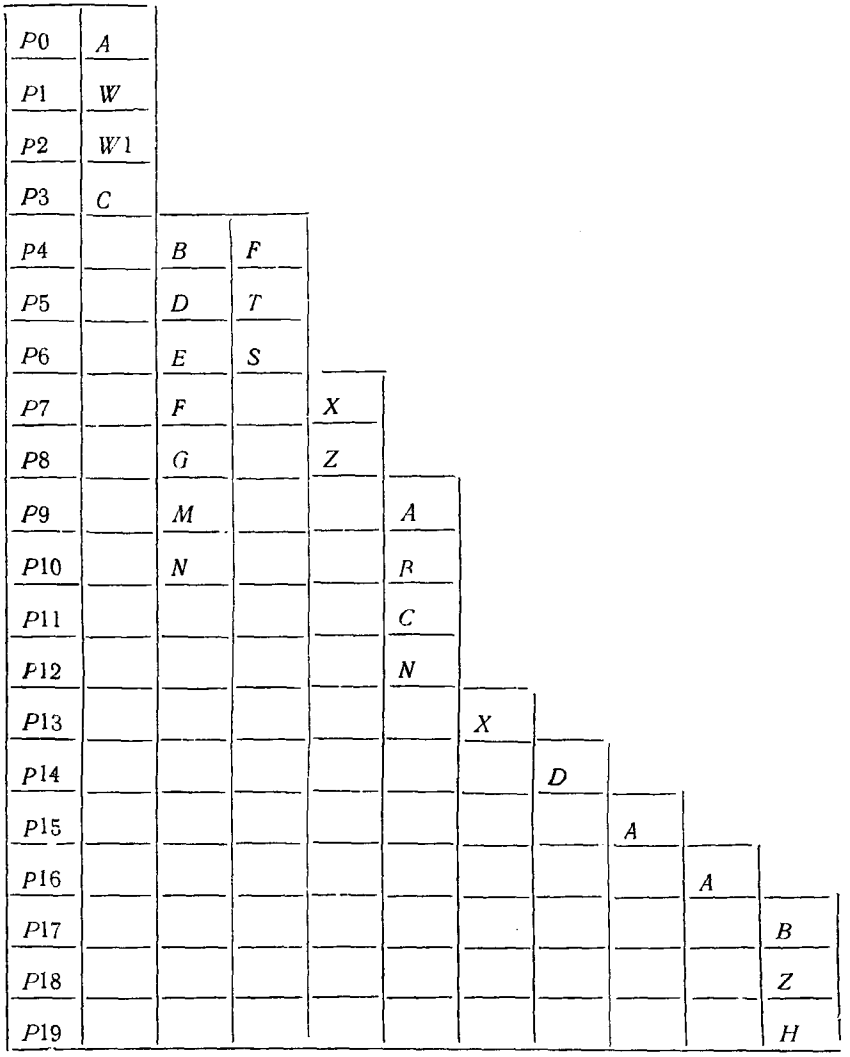
$$n + m + l + \sum_{i=1}^{k} S_i,$$

где S_i — длина участка памяти для величин, локализованных в описании i-й описанной в данном блоке процедуры.

Рассмотрим распределение фиксированной памяти на примере программы, имеющей следующую структуру:

```
M1 : begin real A; array W, W1 [1 ; 100]; integer C; ...
M2 : begin integer B; array D [1 : C];
        real E, F, G, M, N; ...
     end M2; ...
M3 : begin real F;
           real procedure T (X, Y, Z);
           value X, Z; real X; array Z;
           integer Y;
         begin real A, B, C;
           procedure Q (X); value X; real X; ...
           begin integer D; ...
           end;
           integer N; procedure R (Y); real Y;
           begin real A; ...
           end; ...
         end;
         array S [1 : C]; ...
         M4 : begin real A, B; array Z [1 : 5];
           Boolean H; ...
           end M4; ...
   end M3; ...
end M1;
```

Тогда если $P0$ — начальная ячейка фиксированной памяти, то распределение фиксированной памяти для величин данной программы будет иметь вид:



Ячейки $P0$—$P3$ составляют участок блока $M1$. Так как в этом блоке нет описаний процедур, то участок совпадает со своей первой частью. Точно так же ячейки $P4$—$P10$ составляют участок блока $M2$. Так как блок $M2$ является внутренним по отношению к $M1$, то участок блока $M2$ располагается за участком блока $M1$.

Ячейки $P4$—$P15$ составляют участок блока $M3$. Ячейки $P4$—$P6$ составляют первую часть этого участка, ячейки $P7$—$P15$ — вторую часть. Вторая часть состоит из участков двух процедур. Ячейки $P7$—$P14$ — участок процедуры T, ячейка $P15$ — участок процедуры R. Так как блоки $M2$ и $M3$ являются невложенными, то их уча-

стки имеют одно и то же начало. Аналогично можно выделить составляющие участки процедуры T.

В ячейках, соответствующих простым переменным, во время работы готовой программы находятся текущие значения этих переменных. Ячейки, соответствующие процедурам-функциям, служат для помещения туда значений функций.

Каждая ячейка, соответствующая идентификатору массива, во время работы готовой программы содержит в динамической памяти начало вектора информации о массиве. Этот вектор содержит информацию о размерности массива и о текущих значениях нижних и верхних границ граничных пар. За вектором информации о массиве в динамической памяти расположены в лексикографическом порядке компоненты массива.

Вектор информации о массиве размерности n занимает $n+1$ ячейку динамической памяти. В первой из этих ячеек по второму адресу находится размерность массива n.

В $i+1$-й из этих ячеек ($i=1, 2, \ldots, n$) находится текущее значение нижней границы i-й граничной пары и размер i-й позиции индекса, т. е. количество значений, допустимых для индексов в i-й позиции. Размер i-й позиции индекса равен увеличенной на единицу разности между текущими значениями верхней и нижней границ i-й граничной пары.

Например, для массивов, описанных как

array $A, B, C, [1:5, 1:10]$

вектор информации будет иметь вид:

		2
	1	5
	1	10

Вектор информации о массиве используется в готовой программе для определения адресов переменных с индексами.

При входе в блок, в котором описаны некоторые массивы, во время работы готовой программы распределяется память под эти массивы относительно первой свободной ячейки динамической памяти. Это достигается путем обращения к стандартной подпрограмме распределения памяти один раз для каждого сегмента из списка массивов.

Каждому массиву размерности n, состоящему из l компонентов (длины l), отводится в динамической памяти $n+1+l$ ячеек, первые $n+1$ из которых служат для помещения вектора информации о массиве, остальные — для помещения компонентов массива. Длина массива $l=\prod_{i=1}^{n} l_i$, где l_i — размер i-й позиции индекса.

При обращении к подпрограмме распределения памяти для сег-

мента, содержащего mn-мерных массивов, дополнительная информация занимает $n+\text{entier}\ ((m+1)/2)$ ячеек в готовой программе.

i-я ($i=1,\ldots,n$) из первых n ячеек содержит по первому и второму адресам соответственно адреса текущих значений нижней и верхней границ i-й граничной пары. В ячейке, соответствующей i-й граничной паре, в знаковом разряде стоит 1. В остальных ячейках по второму и первому адресам стоят адреса фиксированной памяти, соответствующие идентификаторам массивов в сегменте.

Если m нечетное, то первый адрес последней ячейки пуст. В последней ячейке в знаковом разряде стоит 1.

Подпрограмма распределения памяти, используя n первых ячеек дополнительной информации обращения, строит вектор информации о массивах данного сегмента, вычисляет длину l, которую имеет каждый массив из сегмента. Выбирая по очереди каждый адрес из остальных ячеек дополнительной информации, подпрограмма записывает по этому адресу содержимое ячейки α, т. е. адрес первой свободной ячейки динамической памяти, записывает вектор информации о массиве начиная с первой свободной ячейки динамической памяти (находится в α), содержимое ячейки α увеличивает на $n+1+l$. В результате этих действий будет отведена память под все массивы из сегмента и в соответствующих идентификаторам массивов ячейках фиксированной памяти будет стоять начало вектора информации о данном массиве в динамической памяти.

При выходе из блока память, которую занимали описанные в нем массивы, должна быть освобождена и доступна для дальнейшего использования. Это освобождение могло бы заключаться в том, чтобы при выходе из блока, в котором описаны массивы, восстановить значение α таким, каким оно было при входе в данный блок. Поэтому необходимо, чтобы значение α в момент входа в такой блок сохранялось и было доступно при выходе. Для хранения значения α таким, каким оно было при последнем входе в блок, в котором описаны массивы, используется ячейка β. Значением ячейки β в любой момент является значение α при самом последнем входе в блок, в котором описаны массивы. Динамика управления массивами в каждом отдельном трансляторе имеет свои отличительные особенности.

В процессе трансляции на всех этапах выполняется семантический контроль исходной программы. Ошибки семантики фиксируются транслятором и организуется аварийное продолжение процесса трансляции. После завершения работы транслятора (или невозможности дальнейшего продолжения из-за ошибок) производится разгрузка памяти ошибок по стандартной форме, принятой в данной системе программирования.

Если процесс трансляции закончился успешно, то производится вывод полученной программы в соответствии с заказом. Это может быть, в частности, и вывод на широкую печать в виде листинга (см. главу 5).

9.6. Автоматизация разработки трансляторов

Процесс разработки трансляторов с алгоритмических языков путем написания программ в машинных кодах (или на ассемблерном языке) отличается большими трудовыми затратами. Так, разработка АЛЬФА-транслятора с алгоритмического языка АЛЬФА потребовала по оценке разработчиков 34 человеко-года. На изготовление первого компилятора с языка ФОРТРАН для ВМ-704 потребовалось 18 человеко-лет.

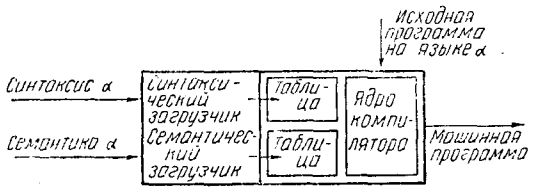


Рис. 9.3. Упрощенная схема компилятора компиляторов

Решение проблемы автоматизации разработки трансляторов может быть найдено на базе использования специального алгоритмического языка для описания процессов трансляции. Среди множества языков, обеспечивающих описание процессов трансляции как процессов символьной обработки информации, можно указать на один из наиболее распространенных в нашей стране алгоритмический язык ЭПСИЛОН. С этого языка разработаны трансляторы для ЭВМ «М-220», «БЭСМ-6» и «Минск-22». Язык обработки символьной информации значительно повышает производительность труда программиста, делает его программы в значительной степени не зависящими от конкретной вычислительной машины.

Системы написания компиляторов, или компиляторы компиляторов, обеспечивают создание компиляторов с алгоритмических языков. Упрощенная схема компилятора компиляторов представлена на рис. 9.3.

Технологический процесс разработки компилятора с заданного языка α составляет следующую систему действий. В синтаксический загрузчик вводится формальный синтаксис языка α, выраженный в синтаксическом метаязыке. Программа синтаксического загрузчика строит на основе синтаксиса языка α специальные таблицы, которые управляют распознаванием и разбором программ на языке α. Следующую группу входных данных составляет семантика языка α, записанная на семантическом метаязыке. Программа семантического загрузчика строит таблицу, содержащую описание значений операторов в α.

Ядро компилятора включает программы обеспечения ввода-вывода, генерирования машинных программ и другие средства, используемые любым компилятором. Ядро компилятора вместе с таблицами синтаксиса и семантики составляет компилятор для языка α.

Полученный компилятор является таблично-управляемым транслятором, основанным на распознавателе, использующем единственный выталкивающий магазин. Каждый элемент в этом основном

магазине состоит из двух машинных слов, одно из которых — для синтаксической конструкции, другое — для семантического описания этой конструкции. Когда распознается конкретная конструкция, семантическая таблица и ее семантическое слово определяют действия, предпринимаемые транслятором.

ВОПРОСЫ К ГЛАВЕ

1. Сущность функции Т-преобразования при трансляции.
2. Назначение и задачи, решаемые блоком лексического анализа.
3. Назначение и задачи, решаемые блоком синтаксического контроля.
4. Назначение и задачи, решаемые блоком генерирования машинных команд.
5. Задачи и способы распределения памяти при трансляции.
6. Перечислите основные требования к транслятору с алгоритмического языка описания обработки экономической информации.
7. Дайте сравнительную оценку методов однократного просмотра при трансляции выражений.
8. Запишите в виде блок-схемы алгоритм работы стека при трансляции простых арифметических и логических выражений.
9. Расскажите алгоритм трансляции условных операторов.
10. Сущность распределения памяти при трансляции.

ЛИТЕРАТУРА

Алгоритмы и алгоритмические языки. Вып. 5. М., ВЦ АН СССР, 1971.

Ледли Р. Программирование и использование вычислительных машин. М., «Мир», 1966.

Ренделл Б., Рассел Л. Реализация АЛГОЛа-60. М., «Мир», 1967.

Современное программирование. Сборник статей. М., «Советское радио», 1966.

Фишер Ф. П., Сундл Д. Ф. Системы программирования. М., «Статистика», 1971.

Флорес А. Программное обеспечение. М., «Мир», 1971.

Хопгуд Ф. Методы компиляции. М., «Мир», 1972.

Языки программирования. М., «Мир», 1972.

РАЗДЕЛ 4

ОПЕРАЦИОННАЯ СИСТЕМА

Глава 10

ОСНОВНЫЕ ПОНЯТИЯ ОПЕРАЦИОННОЙ СИСТЕМЫ

10.1. Введение

Операционная система (ОС) представляет собой совокупность (библиотеку) программ, позволяющих автоматизировать процесс составления проблемных программ и подготовки их к выполнению на машине, осуществляющих управление системой во время ее работы. Часть операционной системы, постоянно находящаяся в основной памяти, называется резидентной частью ОС или ядром системы. Программы, вызываемые в основную память для выполнения определенных функций и не хранящиеся там постоянно, называются транзитными программами или просто транзитами.

Транзиты вызываются в один и тот же участок основной памяти, который называется транзитной областью управляющей программы. Вся область основной памяти, занимаемая ядром и транзитами, называется областью управляющей программы. Остальная часть основной памяти образует область проблемных программ. Разделение управляющей программы на ядро и транзиты позволяет минимизировать область управляющей программы и соответственно увеличить область проблемных программ.

В силу того, что транзиты вызываются на одно и то же место основной памяти, перекрывая друг друга, они называются еще перекрывающими программами.

Операционная система спроектирована для работы в различных режимах. Поэтому на вход системы может одновременно поступить большое количество работ, выполнить которые сразу невозможно просто из-за того, что нет достаточного количества ресурсов. К ресурсам системы относятся центральный процессор, место в основной и вспомогательной памяти, отдельные программы устройства управления вводом-выводом, каналы, таймер, консоль оператора.

Вся работа, поступающая на вход системы, делится на независимые задания, являющиеся единицей работы для операционной системы. Задания описываются с помощью управляющих карт.

В пакетном режиме они группируются для ввода в систему во входном потоке заданий. Поток заданий — совокупность нескольких заданий, декларируемых управляющими операторами, тексты программ и входные массивы данных для отдельных заданий. Основное назначение управляющих операторов — определить и специфировать задание или его часть.

Составная часть задания, выполняемая самостоятельно, называется шагом задания. Шаги одного задания могут влиять на выполнение друг друга. В единицу времени всегда выполняется один шаг одного задания.

Задания во входном потоке описываются с помощью управляющих карт. Управляющие карты представляют собой операторы языка управления заданиями. Язык управления заданиями предназначен для описания заданий и имеет следующие типы операторов: оператор задания, оператор выполнения, оператор описания данных, оператор процедуры, оператор команды, оператор ограничения, пустой оператор.

Оператор задания отличает начало одного задания и конец управляющих операторов предыдущего задания во входном потоке. Оператор имеет несколько параметров, позволяющих прекратить выполнение задания до его завершения при аварийной ситуации, потребовать дополнительный объем основной памяти, задать или изменить приоритет задания.

Оператор выполнения обозначает начало шага задания и идентифицирует программу, которая должна быть выполнена по этому шагу задания. Он также имеет ряд параметров, позволяющих управлять ходом выполнения шага задания.

Оператор описания данных идентифицирует массивы данных и запрашивает распределение ресурсов ввода-вывода.

Оператор процедуры появляется как первый управляющий оператор в каталогизированной процедуре и используется, чтобы приписать недостающие значения символическим параметрам, определенным в процедуре.

Под каталогизированной процедурой понимается следующее. Если некоторая последовательность заданий часто встречается во входном потоке, то можно присвоить имя этой последовательности и оформить ее как каталогизированную процедуру. В дальнейшем эта последовательность заданий хранится в одной из библиотек системы и уже не описывается во входном потоке, где только указывается имя соответствующей ей каталогизированной процедуры. Например, имеется часто повторимая группа заданий: скомпилировать отдельные части программы, скомпоновать рабочую программу, выполнить ее. Вместо того чтобы каждый раз описывать три задания, присвоим их последовательности имя каталогизированной процедуры. Тогда во входном потоке можно будет использовать одну карту с именем этой каталогизированной процедуры вместо множества карт с описанием трех работ.

Оператор команды используется человеком-оператором за пуль-

том машины для выполнения ряда функций по управлению ходом вычислительного процесса.

Операторы ограничения и пустой являются маркерами во входном потоке (например, чтобы отделить данные от последующих управляющих операторов или отметить конец управляющих операторов для некоторых заданий).

Операционная система имеет модульную структуру, которая позволяет потребителю приспособить систему к конкретным конфигурациям технических средств. Отдельные программные компоненты операционной системы, а также конкретные функции управляющей программы могут включаться в систему по желанию пользователя. Процесс создания конкретной операционной системы, учитывающий особенности вычислительной машины и задач пользователя, называется генерацией системы. Средства генерации системы представляют собой совокупность программ и правил, необходимых для выполнения генерации.

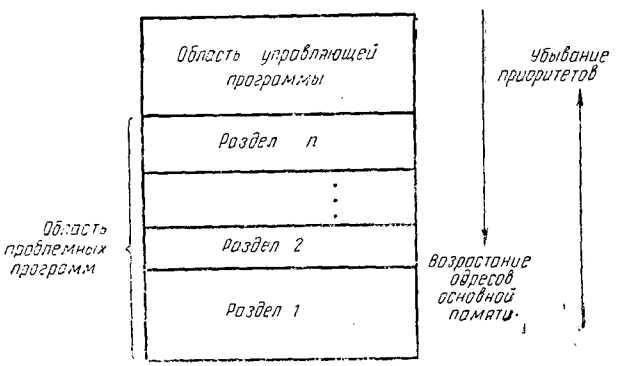


Рис. 10.1. Распределение основной памяти между разделами

Основная память, предоставляемая проблемным программам, заранее при генерации операционной системы делится на фиксированное число разделов. Число таких разделов зависит от конфигурации операционной системы и колеблется от трех для менее мощных конфигураций до пятнадцати для наиболее мощных систем. В каждом разделе одновременно может выполняться одно задание. Программы, выполняемые в разных разделах, не зависят друг от друга в силу независимости заданий.

Для предотвращения возможного разрушения программы, выполняющейся в одном разделе, в результате неверной работы программы другого раздела каждому из разделов присваивается собственный ключ защиты памяти. Область управляющей программы имеет ключ защиты 0.

Между разделами установлен жесткий приоритет. Самый высокий приоритет имеет раздел, расположенный в наиболее высокоадресуемой основной памяти. Самый низкий — раздел, расположенный сразу за областью управляющей программы (рис. 10.1).

Каждому разделу отводится не только участок основной памяти, но и устройства ввода-вывода. Для каждого раздела существует свой входной поток заданий.

С точки зрения потребителя при многопрограммной работе каждый раздел вместе с набором внешних устройств, закрепленных за ним, может рассматриваться как отдельная машина. Будем называть такую «самостоятельную» машину потоком.

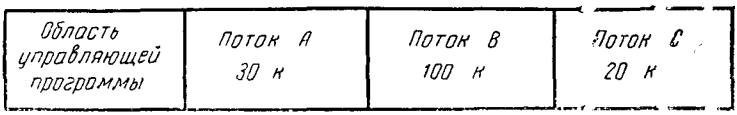


Рис. 10.2. Пример распределения основной памяти между потоками

Понятие потока вводится для более эффективного использования ресурсов вычислительной системы. Так как возможности потоков (объем основной памяти, количество разделов и закрепление внешних устройств) задаются во время генерации операционной системы, потребитель может проанализировать характер выполняемых им на машине работ, более эффективно использовать ресурсы системы. Например, пусть конфигурация машины имеет 256 килобайтов основной памяти, одно устройство чтения с перфокарт, два печатающих устройства, три консоли и другие стандартные устройства. Предполагаемую работу, которую мы собираемся выполнять на машине, можно подразделить на 3 класса: задачи (класс A), которые требуют небольшого объема памяти, перфокарточное устройство и устройство печати; задачи (класс B), которые требуют большого объема памяти и не требуют медленных устройств, задачи (класс C), которые требуют небольшого объема памяти и печатающее устройство. В этом случае наиболее эффективным использованием машины будет разделение ее на три потока A, B, C. Деление памяти между потоками показано на рис. 10.2.

Потоку A будет выделено устройство чтения с перфокарт и печатающее устройство, потоку C — печатающее устройство. В этом случае программы могут работать в индивидуальном потоке и не требовать распределения устройств. Кроме этого, можно лучше управлять загрузкой заданий, используя для этого поток с минимальной памятью, например C, а не поток B, требуя при необходимости перераспределения устройств.

С понятием потока связано понятие ранг, которое играет ту же роль для потоков, что и приоритет для разделов. Ранг позволяет управлять порядком загрузки заданий для выполнения:

0 — задания с этим рангом не будут загружаться, а будут ждать, пока не изменится ранг;

1 — задания будут загружаться последовательно. Если хотя бы одно задание не может быть загружено, то ни одно задание с рангом 1 не будет загружено в этот поток;

2—8 — задания загружаются по мере появления;

9 — этот ранг присваивается заданию, которое не может быть загружено, так как не все требуемые ему устройства свободны или недостаточен объем свободной памяти. Для задания с таким рангом все свободные устройства резервируются до тех пор, пока задание не сможет быть загружено. Аналогично резервируется память.

Как только управляющая программа распознала шаг задания и определила требуемые ему ресурсы, формируется задача. Задача — единица действия в операционной системе. Следует четко различать понятия: задание, задача, программа. Для того чтобы выполнить работу в вычислительной системе, необходимо ввести задание на выполнение этой работы. Например, нужно выполнить в машине программу *А*. Программа *А* написана на одном из алгоритмических языков — АЛГОЛе. Следовательно, сначала должен работать компилятор с языка АЛГОЛ. Скомпилированная программа представлена в объектном коде и еще не готова к выполнению. Работу по подготовке программы к выполнению исполняет программа операционной системы, называемая РЕДАКТОР. Скомпилированную и отредактированную программу можно загружать в основную память и выполнять.

Таким образом, задание на выполнение программы *А* будет состоять из трех шагов: скомпилировать программу *А*, отредактировать, выполнить. Как только операционная система распознает первый шаг задания — скомпилировать программу *А* — и определит требуемые ему ресурсы, будет сформулирована задача, действие которой заключается в выполнении компилятора с языка АЛГОЛ. Программа, которая выполняется для данной задачи, называется подчиненной данной задаче. В нашем примере компилятор с АЛГОЛа подчинен задаче, образованной из первого шага задания.

Задаче, образованной из второго шага задания, — отредактировать — будет подчинена программа РЕДАКТОР. Задаче, образованной из третьего шага задания, подчиняется скомпилированная и отредактированная программа *А*.

Задача является тем объектом операционной системы, которому выделяются ресурсы: место в основной памяти для подчиненной программы, ресурсы ввода-вывода, ключ защиты памяти, время центрального процессора и др. Именно задачи конкурируют между собой за получение времени процессора и других ресурсов системы. Получение задачей времени центрального процессора означает, что все другие ресурсы уже выделены задаче и управление получает программа, подчиненная данной задаче.

Программа, подчиненная задаче, может вызвать другую программу двояко: программа вызывает другую в рамках первоначаль-

ной задачи; программа обращается к другой программе, для которой формируется новая задача. При инициировании шага задания формирование задачи производит операционная система. Под инициированием шага задания будем понимать его распознавание, определение, есть ли в наличии требуемые ему ресурсы и, если они есть, формирование для него задачи. Образование задачи во время вызова новой программы осуществляется путем использования специальной макрокоманды внутри вызывающей программы. Задачи, сформированные операционной системой, могут выполняться независимо и параллельно. Задачи, относящиеся к одному шагу задания и сформированные во время вызова одной программой другой, могут зависеть друг от друга. Эта зависимость отражает логику программ, составленных пользователем.

Основу операционной системы составляет управляющая программа. Ее назначение — автоматизировать управление работой вычислительной машины. Управляющая программа выполняет функции подготовки операционной системы к функционированию, приема заданий и подготовки к их выполнению, управления ходом выполнения задач, управления процедурами ввода-вывода. Эти функции реализуются программами ПЕРВОНАЧАЛЬНАЯ ЗАГРУЗКА, УПРАВЛЕНИЕ ЗАДАНИЯМИ, СУПЕРВИЗОР ЗАДАЧ, СУПЕРВИЗОР ВВОДА-ВЫВОДА, ЛОГИЧЕСКАЯ СИСТЕМА УПРАВЛЕНИЯ ВВОДОМ-ВЫВОДОМ.

Часть управляющей программы, которая организует прием заданий из входного потока заданий, осуществляет их контроль, подготовку к выполнению и запуск, называется УПРАВЛЕНИЕМ ЗАДАНИЯМИ (см. главу 4).

Общее планирование порядка выполнения работ определяет тип мультипрограммирования. Существуют три типа мультипрограммирования, которые определяют три основных конфигурации операционной системы: операционную систему с первичной управляющей программой (PCP); операционную систему, которая обеспечивает мультипрограммирование с фиксированным числом задач (MFT); операционную систему, которая обеспечивает мультипрограммирование с переменным числом задач (MVT).

Режим пакетной обработки реализуется первичной управляющей программой, основное назначение которой — проверить правильность задания, выделить требуемые ему ресурсы, определить программу, которая должна выполняться в данном задании, передать этой программе управление, следить за ходом выполнения программы и выполнять действия, связанные с нормальным или аварийным завершением программы. При этом по желанию программиста допускается совмещение выполнения программы одного задания с операциями ввода-вывода программы того же или другого задания. Например, если занято устройство вывода, результаты размещаются в системной библиотеке. При освобождении устройства вывода управляющая программа выводит результаты на это устройство из системной библиотеки во время выполнения другого задания.

В режиме пакетной обработки задания поступают на обработку в порядке их следования. Это приводит к простаиванию ресурсов системы, которые не используются текущим заданием, что является одним из недостатков режима пакетной обработки.

Дальнейшее развитие пакетной обработки — введение режима пакетной обработки с мультипрограммированием. Этот режим реализуется двумя типами управляющих программ в зависимости от организации планирования прохождения заданий через вычислительную систему.

Мультипрограммирование с фиксированным числом задач характеризуется тем, что область основной памяти, отданная проблемным программам, заранее при создании системы делится на фиксированное число разделов. Каждый раздел предназначен для выполнения одной задачи и имеет свой входной поток заданий. Одновременно могут обрабатываться по одному заданию в каждом разделе.

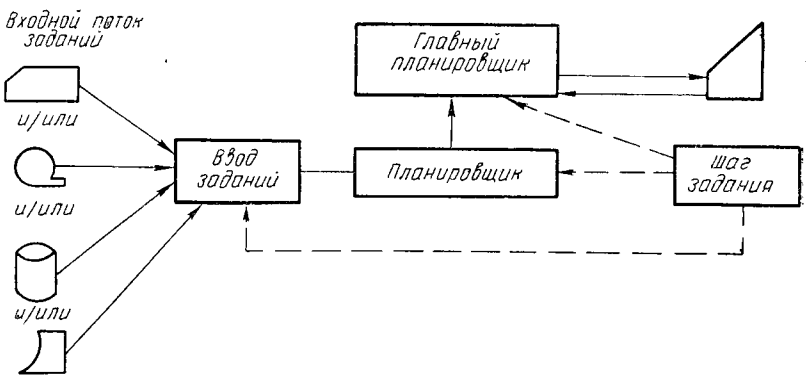


Рис. 10.3. Последовательное планирование

Мультипрограммирование с переменным числом задач является расширением свойств мультипрограммирования с фиксированным числом задач. Отличие заключается в том, что во время выполнения задачи она может порождать другие задачи. В одном разделе одновременно может выполняться несколько задач. В наиболее мощных операционных системах число одновременно выполняемых задач может достигать 52.

Основные положения, излагаемые в книге, справедливы для всех конфигураций системы, если только в тексте не оговорено противное.

Для выполнения своих функций УПРАВЛЕНИЕ ЗАДАНИЯМИ использует информацию, содержащуюся в управляющих операторах языка управления заданиями из входного потока, в директивах, поступающих от оператора, а также пользуется информацией, сформированной другими частями управляющей программы.

УПРАВЛЕНИЕ ЗАДАНИЯМИ представляет собой совокупность программ. В зависимости от организации планирования различают систему последовательного планирования и систему приоритетного планирования. Система приоритетного планирования реализована только в операционной системе с переменным числом задач.

Общая схема системы последовательного планирования изображена на рис. 10.3. Она состоит из программ ВВОД ЗАДАНИЙ, ПЛАНИРОВЩИК, ГЛАВНЫЙ ПЛАНИРОВЩИК. Последовательное планирование заключается в том, что задания из входного потока данного раздела вводятся последовательно. Для каждого задания просматриваются и анализируются управляющие операторы, выделяются устройства ввода-вывода, а оператору выдается сообщение, какие физические тома должны быть установлены. После того как задание получило все требуемые ему ресурсы, оно инициируется как задача, и управление передается СУПЕРВИЗОРу, который осуществляет управление задачами. Управление задаче передает СУПЕРВИЗОР.

Система приоритетного планирования является более мощной (рис. 10.4). Как только задания обнаружены во входном потоке,

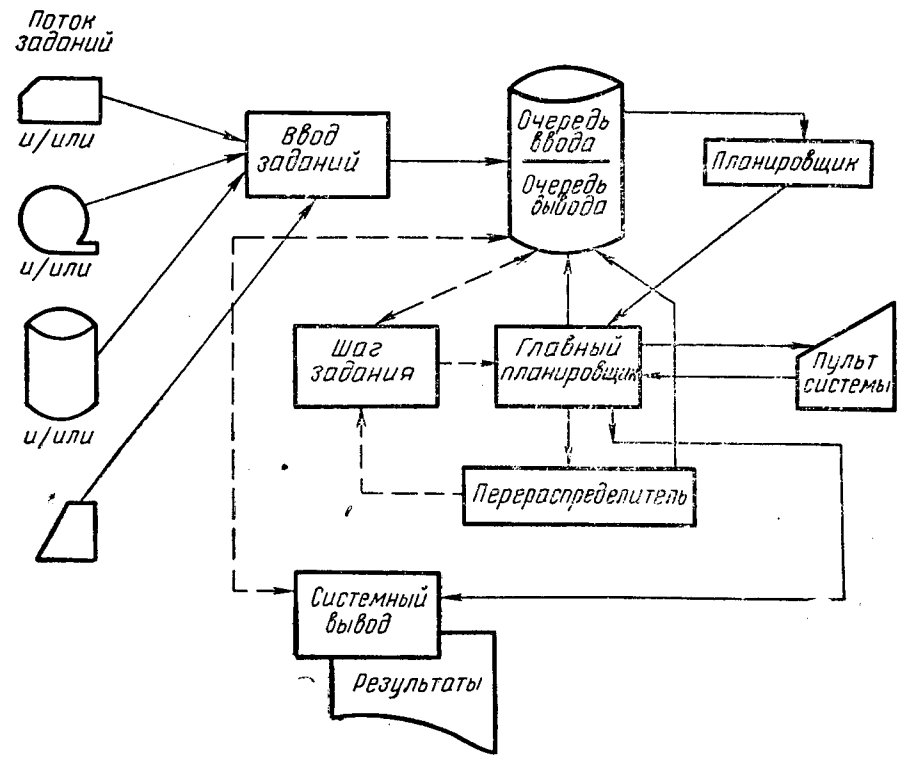


Рис. 10.4. Приоритетное планирование

вся управляющая информация, относящаяся к одному заданию, помещается в устройство с прямым доступом, из которого она в дальнейшем выбирается в зависимости от приоритета задания. Набор управляющей информации называется очередью входных заданий (очередью ввода). Очередь может пополняться из нескольких входных потоков заданий.

Аналогично вместо непосредственной печати или перфорации результатов выходные данные могут быть с большой скоростью размещены в памяти устройства с прямым доступом для более позднего вывода по мере освобождения требуемых устройств. Для этой цели управляющая информация выходных данных каждого задания помещается в устройство с прямым доступом, образуя очередь выходных работ (очередь вывода).

Система приоритетного планирования реализуется программами ВВОД ЗАДАНИЙ, ПЛАНИРОВЩИК, ГЛАВНЫЙ ПЛАНИРОВЩИК, РАСПРЕДЕЛИТЕЛЬ, функции которых расширены по сравнению с системой последовательного планирования, а также добавлены новые программы ПЕРЕРАСПРЕДЕЛИТЕЛЬ, СИСТЕМНЫЙ ВЫВОД.

Система приоритетного планирования более гибка, позволяет более оптимально использовать ресурсы, обеспечивая порядок выполнения заданий в зависимости от их приоритета и наличия ресурсов. Система может также реагировать на задержки, вызываемые установкой или отключением отдельных устройств ввода-вывода, не прерывая выполнения работ.

Система приоритетного планирования обеспечивает ведение журнала учета заданий. Для каждого задания в журнале указывается его имя, время, необходимое для выполнения каждого задания, и другая информация (например, директивы оператора), которая может использоваться для анализа.

УПРАВЛЕНИЕ ЗАДАНИЯМИ обеспечивает также дистанционную пакетную обработку. Дистанционная пакетная обработка означает, что управляющая информация задания может быть передана в систему от дистанционных (удаленных) терминалов. Такая возможность дает удобный и практичный метод совместного использования мощности большой централизованной вычислительной системы, позволяя выполнять задания, поступающие по линиям связи и из входного потока.

Часть управляющей программы, которая контролирует выполнение задачи на всех этапах начиная с момента формирования задачи из задания, введенного для выполнения, и до получения результатов работы, называется УПРАВЛЕНИЕМ ЗАДАЧАМИ. До создания операционных систем различие между программой и задачей было излишне, так как в каждый момент времени программа использовалась только для одной задачи. При мультипрограммном режиме одна и та же программа может использоваться несколькими различными задачами, имеющими свои собственные приоритеты. Поэтому мультипрограммирование означает управление задача-

ми, а не программами. Именно задачи конкурируют между собой за получение времени процессора и других ресурсов системы.

Управление задачами, которые могут выполняться индивидуально или одновременно, осуществляют программы СУПЕРВИЗОРа ЗАДАЧ. СУПЕРВИЗОР включается в работу по сигналам прерывания.

Задачи могут создаваться только двумя способами: путем инициации шага задания или динамическим путем во время вызова одной программы другой. В обоих случаях управляюшая информация о задаче помещается в очередь задач согласно приоритету. Если задача может непосредственно использовать центральный процессор (т. е. другие ресурсы она уже имеет), то она находится в состоянии готовности. Если задача ожидает завершения какой-либо операции, например операции ввода-вывода, то она находится в состоянии ожидания. При работе со многими задачами следует держать центральный процессор постоянно занятым.

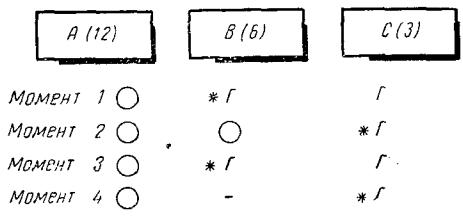


Рис. 10.5. Передача управления между задачами

Первой выполняется задача с самым высоким приоритетом. Однако если задача, которая должна выполняться, находится в состоянии ожидания, то будет выполняться другая задача с более низким приоритетом, но находящаяся в состоянии готовности. Этот процесс иллюстрируется рис. 10.5.

Пусть очередь задач состоит из трех задач: задачи *A* с наивысшим приоритетом 12, задачи *B* с приоритетом 6, задачи *C* с приоритетом 3. В момент времени 1 задача *A* находится в состоянии ожидания, например ждет окончания операции ввода-вывода (изображается О), задачи *B* и *C* — в состоянии готовности (изображается *Г*). Управление передается задаче *B*. Звездочкой отмечены задачи, которые выполняются в данный момент времени.

В момент времени 2 программа, выполняемая для задачи *B*, достигла места, где она должна ждать некоторого события (например, доступа к библиотечной программе). Тогда управление будет передано задаче *C*. В момент 3 задача *B* снова перешла в состояние готовности. Поэтому происходит прерывание выполнения задачи *C*, и управление передается задаче *B*. В момент 4 задача *B* завершается. Управление возвращается задаче *C*.

Прерывание задачи *C* не повлияло на ее выполнение. Всякий раз, когда управление передается от одной задачи к другой, содержимое регистров, слово-состояние программы для задачи запоминается в блоке управления задачей, а при возобновлении ее восста-

навливается. Блок управления задачей представляет собой несколько таблиц, содержащих управляющую информацию задачи.

Работа со многими задачами обеспечивает быстрое время прохождения заданий в режиме пакетной обработки, дает способ управления большим числом телекоммуникационных действий, которые характеризуются множеством задач, большая часть которых находится в состоянии ожидания, позволяет программировать сложные задачи, разбив их на части, которые могут выполняться в системе одновременно, оптимально используя ресурсы и уменьшая общее время решения задач.

10.2. Управление данными

Современное программирование ввода-вывода предоставляет программисту стандартные процедуры для управления данными и обеспечения к ним доступа. Различаются два уровня управления данными: физический и логический. Физический уровень предполагает знание физических внешних устройств и прочих компонентов, необходимых для того, чтобы организовать работу с этими устройствами. Логический уровень предоставляет программисту стандартные процедуры, которые освобождают его от необходимости точного знания конкретных физических устройств. Пользуясь логическим уровнем управления, программист должен заботиться только о логической структуре своих данных.

Каждый уровень управления реализуется совокупностью программ, называемых соответственно физической системой управления вводом-выводом (PIOCS) и логической системой управления вводом-выводом (LIOCS).

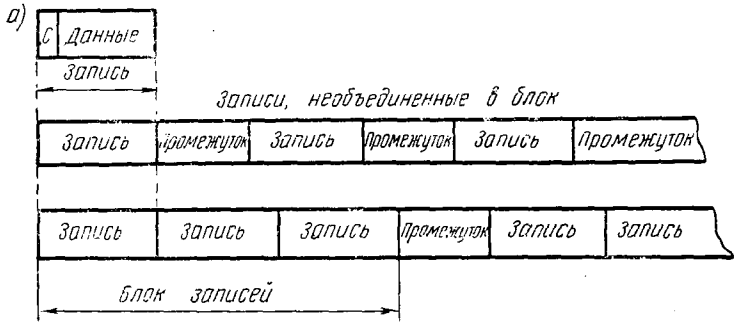
Физическая система управления вводом-выводом используется всеми программами независимо от того, используется ли в них программирование ввода-вывода на логическом уровне или нет. Она контролирует выполнение канальных программ, даже если они составлены в проблемной программе программистом. Физическая система управления осуществляет планирование и установление очередности операций ввода-вывода, контроль выполнения операций, обработку ошибок и прочих исключительных условий, относящихся к устройствам ввода-вывода, обработку прерываний ввода-вывода. Все функции физической системы управления вводом-выводом выполняются программами, которые являются составной частью СУПЕРВИЗОРа.

На логическом уровне основной единицей информации, подлежащей обработке, является логическая запись (или просто запись). Запись представляет собой логический объект, определяемый существом рассматриваемой проблемы. Например, записью может быть информация о заработной плате одного рабочего.

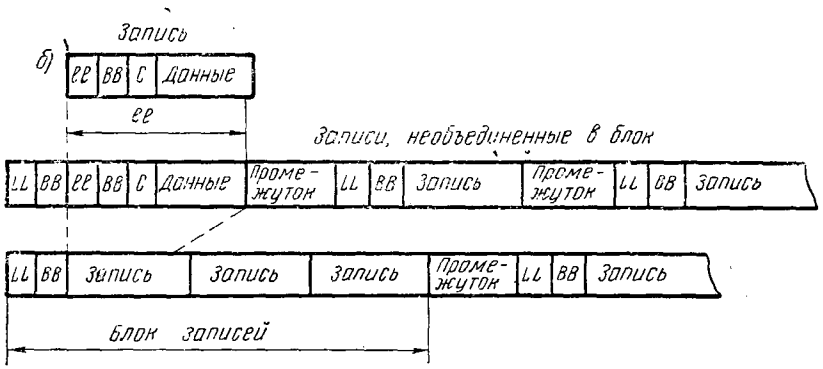
Совокупность записей, объединенных по некоторым общим признакам, образует файл данных. Примерами файлов являются сово-

купность записей, образующих платежную ведомость; программы; очереди сообщений; различные таблицы.

Файл может содержать записи одного из трех форматов: фиксированной, переменной и неопределенной длины. Название формата определяет длина записей, из которых состоит файл. Для некоторых устройств допускаются записи любого формата, например для дисков, магнитных лент. Некоторые устройства допускают определенные форматы, например, для перфокарточных устройств допу-



а) записи с фиксированным форматом



б) записи с переменным форматом

Рис. 10.6. Форматы данных:

с — управляющий знак; *LL* — два байта, в которых хранится длина записи; *ВВ* — два байта, зарезервированные для системного использования; *LL* — два байта, в которых указывается длина блока

скаются только записи фиксированной длины. Неопределенный формат предназначен для обслуживания, например, устройств чтения с перфоленты.

Для повышения эффективности обработки записи файлов на магнитных лентах или дисках могут быть сгруппированы в блоки.

Блокирование записей уменьшает количество операций ввода-вывода, требуемых для обработки файла, и экономит внешнюю память, так как уменьшается число промежутков (рис. 10.6).

Блоки бывают трех типов: фиксированного формата, переменного и неопределенного. Для каждого файла должна быть указана максимальная длина блока *LL*. Тогда блоки фиксированного формата имеют длину, равную *LL*, блоки переменного и неопределенного формата имеют длину не больше *LL*. В отличие от неопределенного блок переменного формата в начале блока содержит поле длины блока.

Блоки фиксированного и переменного форматов могут содержать одну и более записей. Фиксированные блоки содержат записи фиксированной длины. Переменные — записи переменной длины. Для файлов неопределенного формата управление данными не делает различия между блоком и записью. Если все же требуется компоновка или раскомпоновка записей, они должны быть выполнены программой пользователя.

Стандартный блок внешней памяти называется томом. Примерами тома являются катушка магнитной ленты, пакет дисков, барабан. Один том может содержать несколько файлов (многофайловый том), один файл может располагаться на нескольких томах (многотомный файл).

Характеристика файла данных, которые позволяют системе опознать этот файл, содержится в специальных блоках, называемых метками файла. Аналогично характеристики тома записаны в метке тома. Метка файла содержит такую информацию, как имя файла, его границы, формат и другую информацию; метки, которые имеют фиксированную структуру и вполне определенное содержание, называются стандартными. Помимо стандартных могут существовать личные метки, которые обрабатываются программистом, а система логического управления лишь считывает их для обработки или записывает. Файлы на магнитных лентах могут не иметь никаких меток либо иметь только стандартные, личные метки или комбинацию стандартных и личных меток. Файлы на дисках обязаны иметь стандартные метки в совокупности с личными метками или без личных меток.

Файлы данных имеют стандартные метки начала и конца файла. Пользователь, помимо этих меток, может использовать личные метки или вообще не иметь меток. В последнем случае ответственность за правильное установление томов возлагается на оператора.

Метка тома всегда содержит порядковый номер тома и ссылку на оглавление тома (*VTOC*) для устройств с прямым доступом. В оглавлении тома находятся ссылки на метки всех файлов данных, хранящихся в данном томе. Том магнитной ленты является последовательным носителем. Поэтому место файла данных в томе определяется его порядковым номером.

Сохранение информации о том, в каком томе находится отдельный файл, трудоемкая работа и часто служит источником ошибок.

Чтобы облегчить решение этой проблемы, управление данными предусматривает автоматическую каталогизацию файлов. Каталог файлов состоит из списков индексов типа дерева и хранится в устройстве с прямым доступом. Вместе с последним уровнем дерева хранится информация о типе устройства и томе, в котором находится файл, и порядковый номер файла. Обращение к файлу, зарегистрированному в каталоге, можно осуществить, используя только имя файла. Если используется составное имя, то каждое входящее в него простое имя соответствует одному из индексов в каталоге.

Используя каталог, можно идентифицировать совокупности файлов данных, связанных между собой общим внешним названием и той последовательностью во времени, в которой они были зарегистрированы в каталоге. Такие совокупности файлов называются группами поколений. Например, название файла ЦЕХ. ЗАРПЛАТА (0) относится к текущему поколению в группе; название ЦЕХ. ЗАРПЛАТА (—1) относится к предшествующему поколению.

Таким образом, группа поколений включает в себя файлы, входящие в один каталог и имеющие одно название. Если получение нового поколения опирается на использование n предыдущих поколений, то новое поколение можно именовать номером последнего поколения. В этом случае управляющую карту с описанием данных не нужно изменять от просчета к просчету. Число поколений n определяется и задается программистом. Когда новое поколение помещается в каталог, самое старое поколение исключается из него. Имеется также возможность систематически изменять n, начиная с 1 до заданного числа N. Когда N достигнуто, процесс начинается снова с единицы.

Том, содержащий весь каталог или часть, называется управляющим томом. Использование разных управляющих томов для групп родственных файлов данных позволяет перемещать части каталога, соответствующие этим группам, как внутри одной машины, так и между различными машинами.

Управляющий том, в котором хранится операционная система, называется резидентным томом системы.

Поиск файла данных начинается в резидентном томе и продолжается, спускаясь с уровня на уровень, пока не будет найден требуемый том. Если он еще не установлен, выдается сообщение оператору с требованием его установки. Для устройств с прямым доступом продолжение поиска после установки тома осуществляется с метки найденного тома.

Для защиты файла от неавторизованного доступа используется пароль. В метку каждого файла может быть введен признак, указывающий, что доступ к данному файлу, возможен, если с пульта правильно задан пароль. Правильность пароля проверяется макрокомандой OPEN (открыт). По названию файла данных и паролю происходит вход в таблицу паролей. Если пароль указан неверно, выполнение программы прекращается. Сама таблица паролей также защищена и имеет главный пароль. Поэтому обратиться к этой

таблице могут только управляющая программа и некоторые привилегированные программисты.

В отличие от обычного файла системный файл имеет фиксированную организацию на логическом и физическом уровне. На логическом уровне системный файл представляет собой иерархически связанную совокупность элементов, которым присвоено имя, на основании которого происходит обращение к этим элементам. На физическом уровне отражено расположение этих элементов на диске.

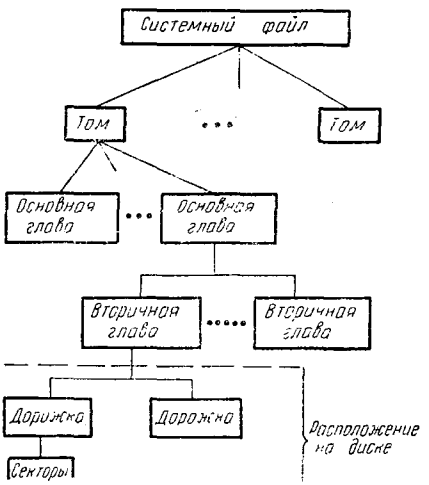


Рис. 10.7. Логическая структура системного файла

На логическом уровне системный файл имеет древовидную структуру, представленную на рис. 10.7. Мы уже знаем, что ОС хранится на системном логическом устройстве SYSRES. Во время функционирования отдельные компоненты системного файла могут располагаться на других системных логических устройствах. Например, библиотека объектных модулей помещается на системное логическое устройство SYSRLR, а ему не обязательно должен быть назначен тот диск, который является резиденцией системы. В общем случае системный файл может размещаться на нескольких системных логических устройствах или томах.

Диску с резиденцией системы, который назначается логическому устройству SYSRES, соответствует том с порядковым номером нуль. Диски, которым назначаются тома с порядковыми номерами, отличными от нуля, могут использоваться для хранения компонентов системного файла, не являющихся резиденцией системы, или в качестве рабочих томов проблемных программ. Порядковый номер тома представляет собой двоичное число, занимающее один байт. Так как в одном байте можно разместить максимальное число 255, то системный файл не должен занимать более 255 томов. Это является несущественным ограничением на размер файла.

В каждом томе располагается одна или несколько основных глав, или разделов, системного файла. Так как термин «раздел» мы используем для обозначения основной памяти, то для обозначения независимой части системного файла, к которой может быть осуществлен доступ по идентификатору, используем термин «основная глава».

Основная глава представляет собой файл специальным образом организованных данных любой природы: программы на исходном языке, языке загрузки или в объектном коде; очередей заданий; совокупности управляющих операторов, описывающих задания; файлы данных; таблицы и другую информацию.

Каждой основной главе присвоено имя, состоящее из шести алфавитно-цифровых символов. Совокупность имени основной главы и порядкового номера тома, на котором эта глава расположена, образует идентификатор основной главы. Формирование идентификатора основной главы производится во время формирования этой главы на диске. Обращение к основной главе можно производить до тех пор, пока диск не будет расформирован. Доступ к основной главе осуществляется по идентификатору.

Назначение и формат основных глав строго фиксированы. Число одновременно функционирующих глав задается пользователем во время генерации системы. При этом пользователь может ввести новые дополнительные основные главы, а также включить все или часть из имеющихся в системе. Рассмотрим назначение наиболее употребительных основных глав.

В операционной системе все программные модули в зависимости от того, в какой стадии подготовки (исходной, объектной или загрузочной) они находятся, могут храниться в соответствующей библиотеке системы. Для хранения программ на исходном языке используется библиотека исходных модулей.

Библиотека исходных модулей представляет собой основную главу, которая хранит в специальном укороченном или сжатом формате программы, написанные на исходном языке. Эти программы будут транслироваться при появлении соответствующего задания на их трансляцию.

Создание основной главы исходных модулей осуществляется с целью ускорения прохождения потока заданий через вычислительную систему. Кроме того, при необходимости повторной трансляции не требуется заново вводить программу на исходном языке, достаточно ввести исправления в исходный модуль и воспользоваться им из библиотеки. Возможность корректировки исходных модулей библиотеки обеспечивает СИСТЕМА ПОДГОТОВКИ ПРОГРАММ.

Результатом работы любого компилятора, входящего в систему, является объектный модуль. Он размещается в библиотеке объектных модулей, которая образует основную главу. Основная глава объектных модулей представляет собой совокупность программ в объектном коде, организованных специальным образом для обработки их СИСТЕМОЙ ПОДГОТОВКИ ПРОГРАММ. Создание основной главы объектных модулей позволяет унифицировать редактирование программ и формирование загрузочного модуля. Это дает возможность разбивать решение большой задачи на самостоятельные части, которые можно программировать на наиболее подходящем для них языке. После трансляции этих частей получен-

ные объектные модули имеют одинаковый формат, не зависят от свойств исходных языков и могут объединяться в один загрузочный модуль программой РЕДАКТОР.

Совокупность загрузочных модулей образует библиотеку загрузочных модулей, которая может храниться в двух основных главах. Загрузочные модули, наиболее часто вызываемые в основную память для выполнения, хранятся в основной главе с библиотекой загрузочных модулей. В основной главе хранятся программы операционной системы, кроме некоторых программ СИСТЕМЫ ПОДГОТОВКИ и компиляторов, а также отлаженные проблемные программы. Наличие этой основной главы обязательно.

Загрузочные модули программ, которые выполняются крайне редко и требуют отладки, размещаются в основной главе с временной библиотекой загрузочных модулей.

Наличие этой основной главы не является обязательным. Она вводится для того, чтобы не перегружать основную главу с библиотекой загрузочных модулей во время длительного функционирования системы.

Размещение отредактированной программы, т. е. загрузочного модуля, в ту или иную основную главу производится РЕДАКТОРом в зависимости от заданного режима работы сформированной программы.

Если программа помещена в основную главу с временной библиотекой, она хранится там до окончания выполнения текущего задания или до того момента, когда в текущем задании потребовано новое выполнение РЕДАКТОРа. В течение этого времени она может вызываться в основную память для выполнения столько раз, сколько потребовал программист. Когда задание окончено или в этом задании получено указание отредактировать следующую программу, программа из основной главы с временной библиотекой стирается и ее уже нельзя вызвать для выполнения иначе, чем начать процесс редактирования сначала.

Если программа хранится в основной главе с библиотекой загрузочных модулей, то она всегда может быть вызвана в основную память для выполнения, не проходя повторного этапа редактирования. Формат обоих глав одинаковый.

Кроме названных библиотек, в системе имеется библиотека макрокоманд ассемблера, которая также образует основную главу. В этой основной главе содержатся макрорасширения макрокоманд ассемблера и сами макрокоманды. При этом все макрокоманды поделены на две группы. Макрокоманды первой группы занимают несколько секторов дорожки. Поэтому их объединяют, и на одной дорожке хранят несколько макрокоманд. Макрокоманды второй группы занимают одну и более дорожек. Каждая из них хранится самостоятельно. Способ хранения макрокоманд накладывает отпечаток на метод доступа к ним.

Библиотека стандартных функций также образует основную главу. Модули стандартных функций хранятся в объектном коде

и вставляются в программы, написанные на языках высокого уровня, с помощью РЕДАКТОРа.

Имеется еще одна основная глава, которая хранит загрузочные модули, используемые СИСТЕМОЙ ПОДГОТОВКИ. В этой главе хранятся загрузочные модули компиляторов, работающих в системе, а также некоторые программы самой СИСТЕМЫ ПОДГОТОВКИ, которые не вошли в основную главу с библиотекой загрузочных модулей. Формат библиотеки совпадает с форматом библиотеки загрузочных модулей. Будем называть ее библиотекой СИСТЕМЫ ПОДГОТОВКИ.

Мы уже знаем, что для ускорения прохождения потока заданий через вычислительную систему, управляющие операторы, описывающие задания, записываются в одну из системных библиотек. Эта библиотека образует основную главу, которая, кроме управляющих операторов задания, содержит очереди заданий, таблицы потоков, таблицу назначения логическим томам физических устройств. С назначением всех этих таблиц мы познакомимся при дальнейшем изучении курса. Будем называть эту главу библиотекой заданий.

Мы также знаем, что для сокращения входного потока заданий используются каталогизированные процедуры. Описания этих процедур, управляющие операторы каталогизированных заданий и их указатели образуют основную главу каталогизированных процедур. Назначение этой главы — обеспечить работу с каталогизированными процедурами.

Чтобы ускорить выполнение некоторых заданий, которые используют исходные файлы данных на перфокарточных устройствах, эти данные размещаются в специальной основной главе вводных данных. Файлы, размещенные в этой главе, имеют последовательную организацию и доступны проблемной программе обычным образом. В этой основной главе вводных данных размещаются небольшие файлы, сформированные отдельными компонентами ОС.

Подобная глава сформирована для хранения выходных данных, журнала системы, очереди выходных работ. Эта глава называется основной главой выходных данных.

Для расширения основной главы выходных данных при организации псевдовывода вводится еще одна основная глава псевдовывода. Если устройство, которое требуется для вывода результатов проблемной программы, занято, то результаты записываются в главу псевдовывода и хранятся там до освобождения требуемого устройства, после чего сама ОС без вмешательства программиста выводит результаты.

В системе может присутствовать также основная глава, хранящая испытательные программы и тесты устройств. Кроме того, потребитель может по желанию вводить новые основные главы. Он может использовать также не все из перечисленных основных глав. Например, потребитель может не использовать основную главу с временной библиотекой загрузочных модулей.

Основная глава — это наиболее крупная единица системного файла. Доступ к основной главе производится с использованием более мелкой единицы — вторичной главы. Каждая основная глава состоит из одной и более вторичных глав. Одна вторичная глава занимает одну и более дорожек диска. Вторичная глава — логическая единица системного файла, предназначенная для обмена между диском и основной памятью.

Природа и структура вторичной главы зависят от назначения основной главы, в состав которой она входит. Например, вторичная глава, входящая в состав основной главы с библиотекой загрузочных модулей, содержит один загрузочный модуль. Вторичной главе библиотеки макрокоманд ассемблера соответствует, например, одна макрокоманда второй группы. Вторичной главе библиотеки объектных модулей соответствует объектный модуль.

Первая по порядку вторичная глава любой основной главы предназначена для хранения указателя вторичных глав. Указатель вторичных глав содержит список всех вторичных глав, входящих в данную основную главу. Доступ ко вторичной главе осуществляется по ее идентификатору. Идентификатор вторичной главы состоит из идентификатора основной главы, в которую она входит, идентификатора программного модуля, который размещается во вторичной главе, некоторой постоянной и текущей информации.

Системный файл размещается на дисках. Чтобы разобраться в структуре системного файла, рассмотрим, что представляют собой дисковые устройства и каков механизм доступа к ним.

Дисковые устройства могут быть со сменными или несменными пакетами дисков. Дисковое устройство 2311 со сменными пакетами дисков состоит из шести дисков, насаженных на вертикальную ось. Каждый диск имеет две рабочие поверхности: верхнюю и нижнюю. Верхняя поверхность верхнего диска и нижняя поверхность нижнего диска не используются для записи. Поэтому пакет дисков имеет десять рабочих поверхностей. Дисковое устройство 2302 с несменными дисками содержит 25 или 50 дисков, соответственно с 46 или 92 рабочими поверхностями.

Рабочая поверхность диска делится на несколько концентрических окружностей, называемых дорожками. Данные записываются последовательно вдоль дорожки бит за битом, по восемь бит в байте. Бит контроля по нечетности, который добавляется к каждому байту в памяти, на диск не записывается. Пакет дисков имеет 200 дорожек на каждой поверхности и три запасные. На каждой дорожке размещается 3625 байтов. Дисковое устройство с несменными дисками имеет 492 дорожки на каждой поверхности и восемь запасных. На каждой дорожке размещается 4984 байта.

Механизм доступа к сменному пакету дисков состоит из пяти рычагов, которые перемещаются вместе как одно целое. Каждый рычаг имеет две головки, предназначенные для записи или чтения.

Всего имеется десять головок, по одной на каждую поверхность. Гребенка рычагов, перемещаясь горизонтально, может находиться

в 203 различных позициях, т. е. одновременно гребенка рычагов осуществляет доступ к дорожкам на всех поверхностях с одним порядковым номером. Например, если гребенка рычагов находится в позиции пятой дорожки, то она может обратиться за один оборот диска ко всем пятым дорожкам на всех десяти поверхностях.

Область, доступная при одном положении механизмов доступа, называется цилиндром. Концепция цилиндров имеет важное значение, так как перемещение механизмов доступа занимает значительную часть времени, необходимого для доступа к данным и их пересылки. Пакет дисков физически состоит из десяти отдельных горизонтальных рабочих поверхностей, на каждой из которых имеется по 203 дорожки, а с точки зрения доступа он состоит из 203 отдельных вертикальных цилиндров по десять дорожек в каждом.

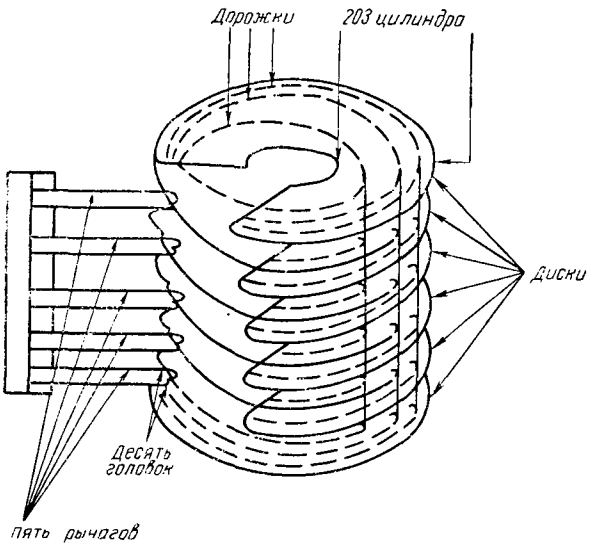


Рис. 10.8. Механизм доступа к пакету дисков

Пакет с несменными дисками имеет две гребенки рычагов для обслуживания 25 дисков и четыре гребенки для обслуживания 50 дисков: одна (или соответственно две) обслуживают 250 дорожек внутренних поверхностей, другая (или две других) — 25 дорожек внешних поверхностей. Каждая гребенка имеет 46 головок для записи или чтения, по одной на каждую рабочую поверхность. Так как механизм доступа за один оборот может обратиться к любой дорожке на всех 46 или 92 поверхностях из 500 имеющихся на поверхности, то несменный пакет дисков имеет 500 цилиндров. Каждый цилиндр имеет 46 или 92 дорожки соответственно числу поверхностей. Механизм доступа к пакету дисков изображен на рис. 10.8.

Таким образом, каждый пакет дисков с точки зрения доступа к нему содержит столько цилиндров, сколько дорожек находится на поверхности диска. Число дорожек цилиндра равно числу рабочих поверхностей диска.

При выборе физической структуры файла время доступа к отдельным компонентам файла должно быть минимальным. Поэтому при размещении системного файла на диске важное значение приобретает концепция цилиндров. Отдельные компоненты системного файла располагаются на дорожках, принадлежащих одному цилиндру, а не на поверхности. При этом любая компонента системного файла занимает несколько дорожек и не требует целого числа цилиндров, хотя практически она занимает целое число цилиндров.

Системный файл представляет собой файл на диске со своим собственным именем и ограниченным сроком использования, равным сроку годности диска. Это постоянно зафиксированный файл, следовательно, любая копия системного диска, полученная с помощью обслуживающих программ, имеет содержание и структуру всех дорожек, как и на основном диске.

Каждая дорожка поделена на несколько секторов. Опыт создания операционных систем показал, что наиболее оптимальным размером сектора является число, кратное 256, т. е. размер сектора D равен $D=256\cdot n$, где $n=1, 2, 3$.

Число секторов на дорожке зависит от длины дорожки и размера сектора. Число секторов на дорожке и его размер всегда фиксированы. В существующих операционных системах размер сектора не кратен 256. Например, для Системы-4 он равен 384 байтам.

Каждая дорожка имеет одну запись, которая называется записью описания дорожки и обозначается $R0$. После записи $R0$ следует сектор с информацией. Длина записи равна восьми байтам. Структура записи представлена в таблице:

Байты	Содержание
0—1	Номер предыдущей дорожки главы или 0
2—3	Номер текущей дорожки главы
4	Резервируется
5—6	Номер следующей дорожки главы или 0
7	Резервируется

Номер дорожки представляет собой полуслово и указывает относительный номер дорожки внутри рассматриваемой компоненты системного файла. Номера предыдущей и следующей дорожек определяют цепочку для связи внутри этой компоненты.

Например, если запись R0 какой-либо вторичной главы содержит информацию 0003000408000640, то мы рассматриваем четвертую текущую дорожку от начала главы, предыдущей является

дорожка три, а следующей — шестая. Все байты любого сектора являются информационными. Структура сектора полностью зависит от назначения главы, которой он принадлежит.

Самая первая дорожка диска с резиденцией системы, дорожка с номером нуль, называется УКАЗАТЕЛЕМ ПАМЯТИ и содержит указатель основных глав и СПИСОК БИТОВ. Ее структура приведена на рис. 10.9.

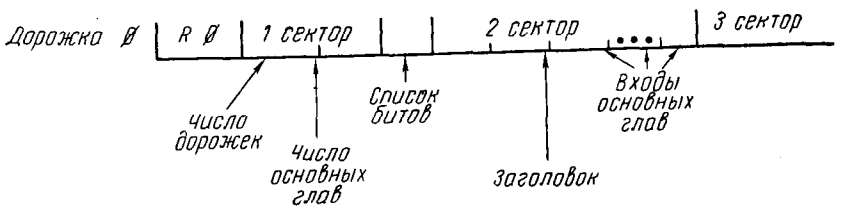


Рис. 10.9. Структура дорожки ∅

Список битов располагается в первом секторе первой дорожки. Каждый бит соответствует одной дорожке на диске с системным файлом. Длина списка бит равна числу дорожек, занятых системным файлом. Любой бит списка равен единице, если соответствующая дорожка уже занята файлом, и нулю, если соответствующая дорожка свободна. Таким образом, имеются сведения о том, какие дорожки диска заняты, какие свободны. Это позволяет расширять главы, используя дорожки данного диска. Список битов имеет следующую структуру: первые два байта содержат число дорожек, отведенных под системный файл, следующие два байта — число основных глав текущего файла. Начиная со следующего байта хранится сам список.

Во втором секторе дорожки с номером нуль хранится указатель основных глав. Он содержит список основных глав, входящих в состав системного файла, вместе с относительными номерами дорожек начала вторичных глав, т. е. с номерами дорожек, хранящих указатель вторичных глав. Структура второго сектора, хранящего указатель основных глав, следующая: байты 0—3 содержат заголовок, далее следуют входы основных глав. Всего может быть до 20 входов. Действительное число входов равно числу основных глав. Структура входа приведена в таблице:

Байты	Содержание
0—5	Имя основной главы
6	Порядковый номер тома
7—8	Номер дорожки первой вторичной главы для данной основной

В каждом входе содержится идентификатор основной главы, состоящий из ее имени и номера тома, и номер дорожки, где хранится указатель вторичных глав, входящих в состав основной главы.

Указатель вторичных глав состоит из заголовка и входов вторичных глав. В заголовке байт 0 содержит нули для любого сектора, кроме последнего. Для последнего сектора байт 0 содержит все единицы. Следующие байты 1—3 содержат нули. Далее следуют входы вторичных глав, структура которых дана ниже.

Байты	Содержание
0—18	Идентификатор вторичной главы
19—20	Относительный номер дорожки
21—26	Резервируются

В каждом входе содержится идентификатор соответствующей вторичной главы и номер дорожки относительно начала основной главы, в которую эта глава входит. Основная глава может иметь до 14 вторичных глав. Это же число ограничивает число входов в указателе вторичных глав.

Связь между отдельными компонентами системного файла осуществляется посредством указателей. В томе с резиденцией системы хранится указатель памяти. Он занимает дорожку с номером нуль и содержит список битов и указатель основных глав.

На основании списка битов можно определить наличие свободных и занятых дорожек в этом томе. На основании указателя основных глав можно определить том, в котором эта глава расположена, и номер дорожки в этом томе, на которой расположен указатель вторичных глав данной основной главы. Номер дорожки тома фактически указывает номер цилиндра, являющегося первым из отведенных данной основной главе.

По указателю вторичных глав можно определить относительный номер дорожки, начиная с которой вторичная глава размещается в томе. Относительный номер дорожки фактически указывает номер дорожки в цилиндре, т. е. номер поверхности диска. Например, основная глава размещается в томе с номером дорожки пять, т. е. ей отведен пятый цилиндр. В этом цилиндре указывается относительный номер дорожки три. Это значит, что указатель вторичных глав физически размещается на третьей поверхности диска на дорожке пять.

Вторичная глава необязательно может занимать смежные дорожки цилиндра, отведенного основной главе. Связь между дорожками, на которых эта вторичная глава располагается, осуществляется на основании цепочки, указанной в записи *R*0, которая имеется на каждой дорожке. В записи *R*0 хранится номер текущей дорожки, номер следующей дорожки, на которой расположена вторичная глава, и номер предыдущей дорожки, если она есть. Тем самым лег-

ко определить местонахождение любой вторичной главы и осуществить к ней доступ.

Следовательно, системный файл имеет специфическую структуру, отличную от структуры обычных файлов, используемых в проблемных программах.

Системный файл подготавливается заранее с помощью обслуживающей программы ФОРМИРОВАНИЕ СИСТЕМНОГО ФАЙЛА. Это программа создает преформат (скелет) системного файла на диске. Для нее указывается число основных глав, размер, тома, на которых они располагаются, и некоторая другая информация.

Файлы могут быть организованы одним из пяти способов: последовательная организация, индексно-последовательная, прямая, частично-последовательная, телекоммуникационная организация.

Последовательная организация подобна ленточной организации, где за предыдущей записью следует последующая. Поэтому, зная местоположение текущей записи, всегда можно определить местоположение следующей записи. Последовательная организация используется для магнитных лент, перфолент, перфокарт, вывода на печать. Она может быть использована и для устройств с прямым доступом. Последовательная организация эффективна в том случае, когда требуется обрабатывать большинство записей файла.

При индексно-последовательной организации записи располагаются в логической последовательности на дорожках устройств с прямым доступом согласно ключу, который является составной частью записи. При этом имеется набор индексов (или отдельный индекс), которые хранятся в таблице индексов и указывают местоположение некоторых основных записей. К любой записи можно обратиться, используя последовательный или прямой метод доступа.

При создании файла с индексно-последовательной организацией на диске выделяется область, предназначенная для добавления новых записей в этот файл. Это позволяет избежать перезаписи всего файла, которая обычно требуется при добавлении записей файла, имеющему последовательную организацию. Хотя добавляемые записи физически не будут располагаться последовательно, обрабатываться они будут в логической последовательности ко ключу.

Прямая организация также предназначена для файлов, которые находятся в томах с прямым доступом. Записи внутри набора данных могут быть организованы любым способом по усмотрению программиста. Записи запоминаются и читаются непосредственно при помощи адресации, указанной программистом. В отличие от индексно-последовательной организации в прямой организации отсутствуют таблицы индексов. Этот метод имеет большую гибкость. Однако, хотя система обеспечивает файл программами для считывания и записи данных, программист несет всю ответственность за логику и программирование, требуемые для определения местонахождения записей, так как именно он устанавливает зависимость между ключом записи и ее адресом на дисках.

Частично-последовательная организация обладает свойствами как последовательной, так и индексно-последовательной организаций. Независимая группа последовательно организованных данных, называемая основной главой, хранится в томе с прямым доступом. Каждая основная глава имеет имя, которое хранится в справочнике. Справочник является частью набора данных и содержит местоположение начала основной главы. Примером использования данного метода является хранение программ. К частично-последовательным наборам данных часто обращаются как к библиотекам.

Телекоммуникационная организация используется только для организации данных, принимаемых и выдаваемых удаленными терминалами. Метод предусматривает прием (выдачу) данных, формирование очередей, поиск в очередях. Очереди могут находиться в основной памяти или во вспомогательной на томах с прямым доступом.

Чтобы осуществить обращение к данным, организованным любым способом, используются языки доступа. Базисный язык доступа обеспечивает управление устройствами без автоматической буферизации и блокирования. Это сравнительно простой язык, позволяющий программисту использовать метод логических устройств. Он также может служить основой для любых методов буферизации и блокирования, созданных программистом.

Язык доступа с очередями предназначен для обеспечения полного набора возможностей буферизации и блокирования в сочетании с максимальной простотой программирования. Он применяется только к организациям данных с последовательными характеристиками.

Совокупность метода организации файла и языка доступа образует определенный метод доступа к этому файлу. Например, дисковая операционная система ЕС ЭВМ имеет три метода доступа: последовательный, прямой и индексно-последовательный. Операционная система ИБМ/360 имеет пять базисных методов доступа и три доступа с очередями.

Методы доступа с очередями позволяют выполнять буферизацию автоматически. Буферизация выполняется, чтобы совместить операции ввода-вывода и обработку для файлов, имеющих последовательную организацию. Если необходимо, управление данными предупреждает запросы на ввод и задерживает запросы на вывод.

Буфер-область основной памяти используется для ввода-вывода. Часть буфера, предназначенная для хранения одной записи, называется сегментом буфера. Группа буферов в области памяти, структура которой определена системой, называется буферным пулом. Каждому файлу отводится буфер из пула, связанного с этим файлом. Если программист не отвел файлу данных буферный пул, ОС выполнит это сама. Если размер буфера не задан, то он устанавливается равным максимальной длине блока.

Запись можно обрабатывать непосредственно в буфере, можно переслать ее для обработки в рабочую область. Управление дан-

ными предусматривает три режима передачи записей для обработки. Эти режимы реализуются макрокомандами ассемблера.

В режиме пересылки каждая запись пересылается из входного буфера в рабочую область и после обработки из рабочей области — в выходной буфер. В этом случае совмещаются операции: пока идет наполнение входного буфера — запись перемещается из рабочей области в выходной буфер; затем происходит заполнение рабочей области из входного буфера и перемещение записи из выходного буфера.

В режиме указания запись никуда не пересылается. Программе пользователя становится доступным указатель на сегмент, в котором находится обрабатываемая запись. Этот режим наиболее удобен для случая, когда обмен между основной памятью и носителем осуществляется только поблочно.

В режиме подстановки, кроме буфера, предусматривается рабочая область, равная длине записи. В этом случае, как и в режиме указания, проблемной программе становится доступным указатель сегмента, в котором находится запись. Буфер и рабочая область периодически меняются местами. Если запись находится в рабочей области, то происходит заполнение буфера. Затем эта запись обрабатывается прямо в буфере, а в рабочую область перемещается следующая запись.

Управление данными обеспечивает эффективные методы отведения буферов. Наиболее общим методом является простая буферизация. Она заключается в том, что каждому файлу отводится один или несколько буферов. Для файлов с фиксированной длиной записей можно использовать обменную буферизацию. Она использует механизм цепочек для эффективного выполнения операций по сбору записей, разбросанных по памяти, или, наоборот, раскидывание их по памяти. При этом имеются в виду аппаратные средства, позволяющие в командах канала определить цепочку по данным.

При помощи цепочки по данным из разрозненных областей основной памяти можно скомплектовать один физический блок для обмена с устройством ввода-вывода. Буферные сегменты, относящиеся к входному набору данных, меняются ролями с рабочей областью или буферными сегментами, относящимися к выходному набору данных. Каждый буферный сегмент может поочередно обрабатываться как входная область, рабочая область и выходная область, а за счет цепочек можно связывать в один блок несколько несмежных сегментов. Обменная буферизация полезна при обновлении файлов с последовательной организацией, при выполнении сортировки слиянием.

Буферизация с цепочками сегментов используется для наборов данных с переменной длиной записей. Сегменты создаются динамически и механизм цепочек используется для связи физически разрозненных сегментов. Метод используется, чтобы не распределять статически память для данных, принимаемых от удаленных терми-

налов (в системе может быть много терминалов, но только часть из них обычно используется одновременно).

10.3. Метод логических устройств

В вычислительных системах третьего поколения к процессору могут подключаться устройства самых разнообразных типов, такие, как ввод и вывод перфокарт, магнитные ленты, диски и т. п.

Каждое внешнее устройство, подключенное к процессору, имеет свой адрес. Адрес указывается обычно в шестнадцатиричной системе счисления в виде *X'сии'*, где *с* — номер канала, к которому подключено это устройство, а *ии* — номер устройства в канале. Внешнее устройство, характеризуемое типом и адресом, назовем физическим устройством. Каждая конкретная вычислительная машина может иметь свой особый набор физических устройств. Количество устройств, а также типы и адреса внешних устройств одной машины не совпадают с набором физических устройств на другой машине.

Для того чтобы сделать программы ОС и проблемные программы независимыми от набора физических устройств, подключенных к конкретной машине, в ОС используется метод логических устройств, который заключается в следующем. Установлен стандартный набор логических устройств, к которым программист обращается в исходной программе при помощи символических имен. Само логическое устройство не определяет ни конкретного адреса физического устройства, ни даже в большинстве случаев его типа.

К началу выполнения программы должно быть установлено полное соответствие между используемыми в программе логическими устройствами и необходимыми физическими устройствами. Это соответствие может устанавливаться во время генерации системы оператором в любое время до начала выполнения программы, программистом непосредственно перед выполнением его программы. Установление соответствия называется процедурой назначения логическому устройству физического устройства. Одно и то же физическое устройство может быть назначено несколькими логическими устройствами.

Все логические устройства делятся на две группы: системные логические устройства и логические устройства программиста. Системные логические устройства используются программами ОС. Логические устройства программиста предназначены для использования в проблемных программах.

Рассмотрим некоторые системные логические устройства, имеющиеся в ДОС ЕС ЭВМ:

SYSRDR — устройство, предназначенное для приема входного потока заданий. Этому логическому устройству могут быть назначены следующие физические устройства: ввод перфокарт, магнитная лента, диск.

SYSIPT — устройство, используемое для ввода данных. Этому

логическому устройству могут быть назначены: ввод перфокарт, магнитная лента, диск.

SYSPCH — устройство, используемое для вывода информации на перфокарты. Этому логическому устройству могут быть назначены: вывод перфокарт, магнитная лента, диск.

SYSLST — устройство для вывода информации на печать. Этому логическому устройству могут быть назначены: устройство печати, магнитная лента, диск.

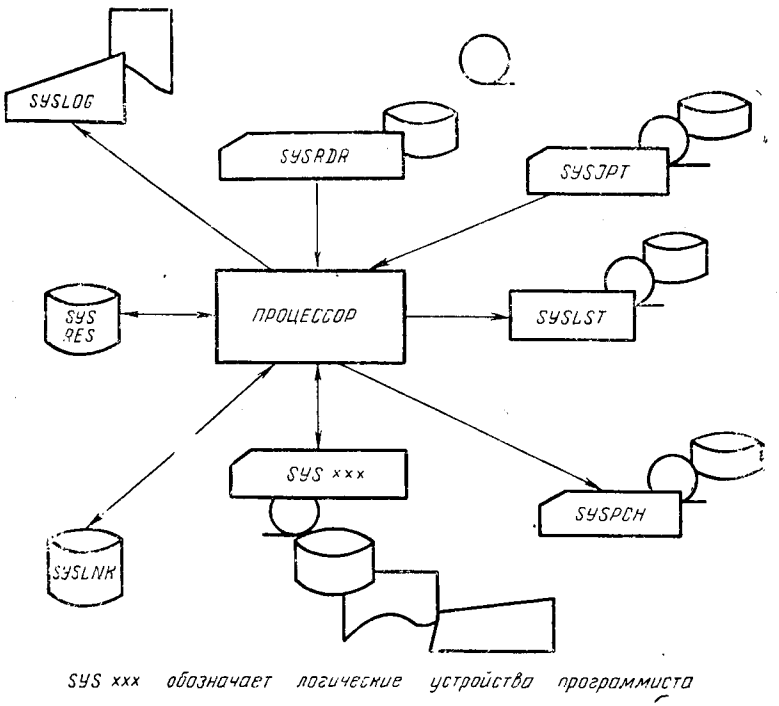


Рис. 10.10. Возможные назначения логическим устройствам физических устройств

SYSLOG — устройство связи с оператором. Этому логическому устройству назначается пультовая пишущая машинка, иногда устройство печати.

SYSRES — устройство, используемое в качестве резиденции системы. Соответствующим физическим устройством может быть только диск.

SYSRLB — устройство для хранения личной библиотеки объектных модулей. Соответствующим физическим устройством может быть только диск. Аналогично для личной библиотеки исходных модулей используется *SYSSLB*. На рис. 10.10 приведены возможные назначения физических устройств логическим.

Каждый поток имеет свой набор системных логических устройств и логических устройств программиста. Однако общее количество логических устройств всех потоков не должно превышать 255. Одно и то же физическое устройство, за исключением диска, не может быть одновременно назначено логическим устройствам разных потоков.

10.4. Общая схема функционирования

Нормальное функционирование вычислительной системы возможно только после того, как она приведена в состояние готовности к работе. Для этого необходимо выполнять ряд действий. На один из механизмов диска устанавливается том с резиденцией системы. Путем нажатия специальной клавиши на пульте машины выполняется процедура первоначальной загрузки, во время которой в основную память из резидентного тома вызывается программа ПЕРВОНАЧАЛЬНАЯ ЗАГРУЗКА. Эта программа загружает в область управляющей программы основной памяти ядро СУПЕРВИЗОРа. После выполнения процедуры первоначальной загрузки система готова к функционированию, т. е. может принимать задания на выполнение проблемных программ потребителя.

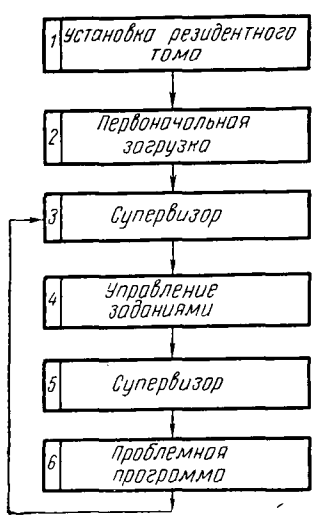


Рис. 10.11. Общая схема подготовки и функционирования системы

Дальнейшее функционирование системы осуществляется под управлением СУПЕРВИЗОРа и заключается в выполнении следующих действий. СУПЕРВИЗОР загружает в область проблемных программ основной памяти программу УПРАВЛЕНИЕ ЗАДАНИЯМИ, которая вводит из пакета заданий управляющие операторы первого (или очередного) задания. На основании информации, поставляемой этими операторами, УПРАВЛЕНИЕ ЗАДАНИЯМИ формирует задачу, которая должна быть выполнена. Завершив свои действия, программа УПРАВЛЕНИЕ ЗАДАНИЯМИ обращается к СУПЕРВИЗОРу с требованием вызвать в основную память программу, указанную в принятом задании. СУПЕРВИЗОР вызывает в основную память эту программу и передает управление для выполнения. Как правило, вызванная программа размещается на месте программы УПРАВЛЕНИЕ ЗАДАНИЯМИ. После окончания выполнения программы, подчиненной данной задаче, управление получает СУПЕРВИЗОР, который вновь вызывает в область проблемных программ управление заданиями для приема управляющих операторов следующего задания.

Схема подготовки и функционирования системы проиллюстрирована на рис. 10.11.

В режиме пакетной обработки задания поступают на обработку из входного потока. Задание может состоять из одного или нескольких шагов. Шаги одного задания выполняются последовательно друг за другом и выполнение каждого следующего шага задания может зависеть от успешного выполнения одного или нескольких предшествующих шагов. Схема организации приведена на рис. 10.12.

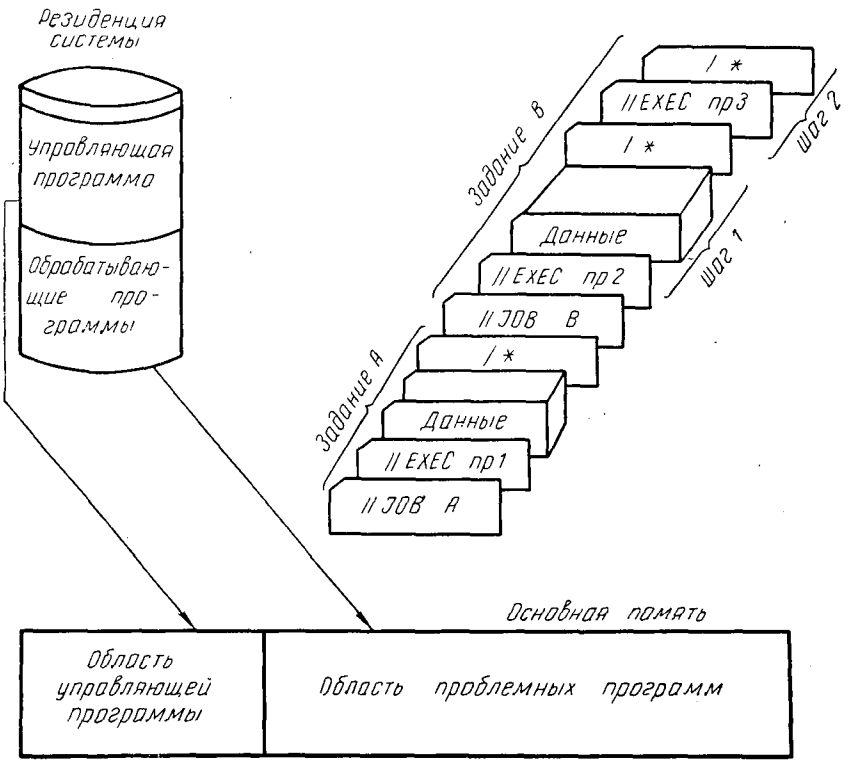


Рис. 10.12. Схема организации пакетной обработки

В режиме пакетной обработки с мультипрограммированием может одновременно вводиться несколько заданий. Первым вводится задание в раздел с наивысшим приоритетом. Как только задача, инициированная в этом разделе, потеряет активность, начнет вводиться задание в раздел с более низким приоритетом.

При распределении времени центрального процессора СУПЕРВИЗОР просматривает разделы в порядке убывания их приоритета и передает управление программе, подчиненной первой встреченной

готовой задаче. Эта программа будет выполняться до тех пор, пока либо не станет готовой задача в более приоритетном разделе и тогда СУПЕРВИЗОР передаст управление ей, либо задача, программа которой выполнялась, потеряет активность, т. е. возникнет ситуация, когда эта задача станет ожидать наступления некоторого события. В последнем случае управление может получить задача менее приоритетного раздела. Если готовых задач в данный момент не оказалось, СУПЕРВИЗОР переводит систему в состояние ожидания. Описанная схема работы СУПЕРВИЗОРа иллюстрируется для работы с тремя задачами рис. 10.13.

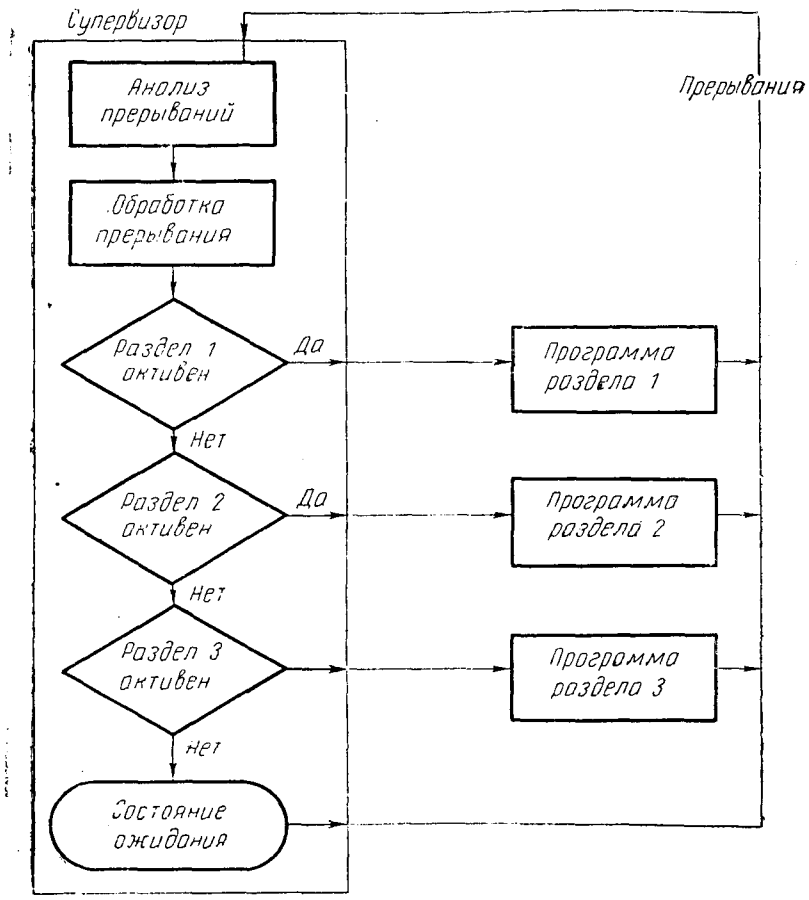


Рис. 10.13. Схема работы СУПЕРВИЗОРа при мультипрограммировании с фиксированным числом задач

ВОПРОСЫ К ГЛАВЕ

1. Понятие раздела и потока.
2. В чем заключается метод логических устройств?
3. Как организуется защита памяти при мультипрограммном режиме?
4. Условия активности задачи.
5. Какой раздел обладает наивысшим приоритетом?
6. В каком состоянии процессора выполняются проблемные программы?
7. В каком состоянии процессора выполняются управляющие программы?
8. Перечислите основные главы и сформулируйте их назначение.
9. Приведите логическую структуру системного файла.
10. Дайте понятие дорожки, цилиндра и поверхности.

Глава 11

СУПЕРВИЗОР

11.1. Назначение и структура супервизора

СУПЕРВИЗОР — часть управляющей программы, которая контролирует выполнение проблемной программы начиная с момента ввода задания на ее выполнение и до получения результатов. СУПЕРВИЗОР получает управление при возникновении сигналов прерывания. Поэтому основной функцией СУПЕРВИЗОРа является обработка прерываний.

СУПЕРВИЗОР производит анализ прерываний, обработку прерываний, поступающих с консоли оператора и прерываний от часов, управление всеми операциями ввода-вывода, распознавание ошибок устройств ввода-вывода, осуществляет ведение системного журнала, управляет работой транзитных программ, а также содержит некоторые программы методов доступа к системному файлу.

СУПЕРВИЗОР имеет модульную структуру и хранится в библиотеке загрузочных модулей в резиденции системы. Во время функционирования системы в основной памяти постоянно присутствует ядро СУПЕРВИЗОРа. В ядро СУПЕРВИЗОРа включены программы, реализующие наиболее часто используемые функции. Остальные программы — транзиты СУПЕРВИЗОРа — реализуют все прочие функции и вызываются в основную память по мере необходимости. Вызов транзитов производится ядром СУПЕРВИЗОРа. Вызов самого ядра и передача ему управления производится программой ПЕРВОНАЧАЛЬНАЯ ЗАГРУЗКА.

Модульность СУПЕРВИЗОРа дает возможность его настройки на конкретную конфигурацию машины. При генерации СУПЕРВИЗОРа потребитель может указать, какие функции должны выполняться СУПЕРВИЗОРом и способы выполнения некоторых из этих функций.

Генерация СУПЕРВИЗОРа является одним из этапов генерации операционной системы. С помощью набора макрокоманд генерации

СУПЕРВИЗОРа потребитель описывает необходимые ему свойства СУПЕРВИЗОРа. После трансляции этих макрокоманд АССЕМБЛЕРом получается конкретный, затребованный потребителем вариант СУПЕРВИЗОРа, который редактируется, каталогизируется и размещается в библиотеке загрузочных модулей. Во время генерации СУПЕРВИЗОРа потребитель может, например, указать режим мультипрограммирования, описать имеющийся набор устройств ввода-вывода, их количество и типы, описать специфические черты оборудования, которые должны быть учтены СУПЕРВИЗОРом.

Введем несколько определений. Мы уже знаем, что процессор может функционировать в одном из четырех независимых состояний. В состоянии $P1$ выполняются обрабатывающие программы и небольшая часть управляющих программ. Будем для краткости называть их программами $P1$. В состоянии процессора $P2$ выполняются действия, связанные с обработкой прерываний. Будем называть программы, реализующие эти действия, программами $P2$.

При возникновении любого прерывания процессор переключается автоматически в состояние $P3$, кроме прерываний от схем контроля машины. В этом состоянии процессор выполняет программы распознавания прерываний и выбора соответствующей программы $P2$ обработки данного прерывания. Будем называть программы, выполняемые в состоянии процессора $P3$, программами $P3$.

При появлении прерываний от схем контроля машины процессор автоматически переключается в состояние $P4$. В этом состоянии выполняются программы обработки прерываний от схем контроля машины, которые заканчиваются остановом машины. Для продолжения вычислений необходимо повторить работу программы ПЕРВОНАЧАЛЬНАЯ ЗАГРУЗКА для восстановления ядра СУПЕРВИЗОРА в основной памяти.

Теперь мы можем ввести новое определение СУПЕРВИЗОРа с точки зрения функционирования процессора. СУПЕРВИЗОРом называется группа программ, выполняемых в состоянии процессора $P2$, $P3$, $P4$. Другими словами, к СУПЕРВИЗОРу относятся все программы $P2$, $P3$, $P4$.

Назначение этих программ мы уже определяли. Программы $P3$ предназначены для управления и выбора функций СУПЕРВИЗОРа, которые требуются в текущий момент времени. Программы $P2$ предназначены для выполнения действий, определенных программами $P3$.

Высокая степень модульности СУПЕРВИЗОРа, когда различные его функции реализуются отдельными независимыми программами, позволяет гибко менять состав программ $P2$. Программы $P2$, реализующие эти функции, могут легко быть добавлены или вычеркнуты, а существующие функции могут быть модифицированы или заменены, не оказывая никакого влияния на немодифицируемые программы.

СУПЕРВИЗОР вызывается в область управляющей программы, являющейся низкоадресуемой основной памятью. В эту область

вызываются резидентные программы СУПЕРВИЗОРа, которые составляют его ядро, и транзитные программы. В состав ядра входят все программы *Р3*, управляющие таблицы и те программы *Р2*, которые были определены во время генерации СУПЕРВИЗОРа как резидентные. Все программы *Р3* всегда являются резидентными.

Программы *Р2*, которые не попали в ядро СУПЕРВИЗОРа, являются транзитными. Они вызываются в основную память по требованию *Р3*, *Р1* и резидентных программ *Р2*. Транзитным программам *Р2* отведена часть области управляющей программы, следующая за областью, отведенной ядру СУПЕРВИЗОРа. Схема распределения области управляющей программы приведена на рис. 11.1.

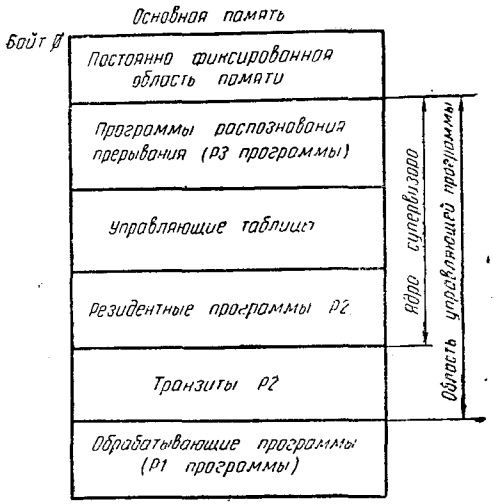


Рис. 11.1. Схема распределения области управляющей программы

С точки зрения выполняемых функций все программы СУПЕРВИЗОРа можно разбить на две группы: управление задачами и управление вводом-выводом. Программы управления задачами предназначены для анализа прерываний, формирования задач, выбора задачи, которой необходимо передать управление. Они управляют всеми действиями системы и называются СУПЕРВИЗОРом ЗАДАЧ. Программы управления вводом-выводом получают управление при появлении прерываний ввода-вывода и предназначены для обработки этих прерываний.

Таким образом, какие бы функции не выполнял СУПЕРВИЗОР, основное его назначение — «узнать» прерывание и обработать его. Поэтому прежде чем перейти к рассмотрению отдельных компонент СУПЕРВИЗОРа, изучим его реакцию при появлении тех или иных прерываний.

11.2. Распознавание прерываний

Некоторые сигналы, поступающие от внешних устройств, а также некоторые ситуации, возникающие в процессоре, могут вызвать прекращение выполнения программы, во время работы которой эти сигналы или ситуации возникли. Такое прекращение выполнения текущей программы называется прерыванием, а сигнал или ситуация, вызвавшая это прекращение, — причиной прерывания. Все

причины прерываний можно разделить на пять классов: обращение к СУПЕРВИЗОРу, программный сбой, машинный сбой, прерывание ввода-вывода, внешнее прерывание.

Прерывание по обращению к СУПЕРВИЗОРу возникает в том случае, когда при выполнении программы встречается команда «Обращение к СУПЕРВИЗОРу». Мнемоническое обозначение этой команды на ассемблере — SVC. Эта команда используется как программами операционной системы, так и обрабатывающими программами в случаях, если, например, требуется выполнить операцию ввода-вывода, загрузить очередной сегмент программы в основную память, открыть или закрыть файл, сообщить СУПЕРВИЗОРу о том, что программа завершена. В общем случае команда «Обращение к СУПЕРВИЗОРу» используется для того, чтобы запросить выполнение функций, обеспечиваемых СУПЕРВИЗОРом. Указание, какая из функций требуется, содержится в самой команде SVC в качестве ее параметра.

Программы операционной системы могут использовать команду SVC в явном виде. В обрабатывающих программах, написанных на ассемблере, используется набор макрокоманд, с помощью которых можно запросить требуемую функцию СУПЕРВИЗОРа. Макрорасширение каждой из этих макрокоманд содержит команду SVC с нужным параметром.

По прерываниям, вызванным программными сбоями, СУПЕРВИЗОР либо передает управление самой проблемной программе для обработки программного сбоя, либо прекращает выполнение задания, для которого эта программа выполнялась. Управление будет передано проблемной программе, если до появления сбоя СУПЕРВИЗОРу с помощью макрокоманды STXIT было сообщено о наличии в проблемной программе подпрограммы обработки сбоев. Макрорасширение макрокоманды STXIT содержит команду «Обращение к СУПЕРВИЗОРу». STXIT предназначена для того, чтобы указать СУПЕРВИЗОРу, что в проблемной программе имеется подпрограмма обработки программных сбоев, связи с оператором или прерываний по таймеру и чтобы при возникновении таких прерываний СУПЕРВИЗОР передавал управление этой подпрограмме. Связь с данной подпрограммой прекращается по завершению выполнения проблемной программы или при выполнении новой макрокоманды STXIT. Получив управление от СУПЕРВИЗОРа и выполнив обработку программного сбоя, подпрограмма может вернуть управление в точку прерывания, использовав макрокоманду EXIT.

Схема обработки прерываний по программным сбоям приведена на рис. 11.2.

Если СУПЕРВИЗОР не получил сообщения о том, что в проблемной программе есть подпрограмма обработки сбоев, то при их появлении он снимает задание, для которого выполнялась эта программа. При этом СУПЕРВИЗОР может произвести распечатку основной памяти, отведенной задаче, и содержимого регистров.

Особенностью прерываний по программному сбою является то, что некоторые из этих прерываний могут запрещаться проблемной программой с помощью команды «Установить маску программы».

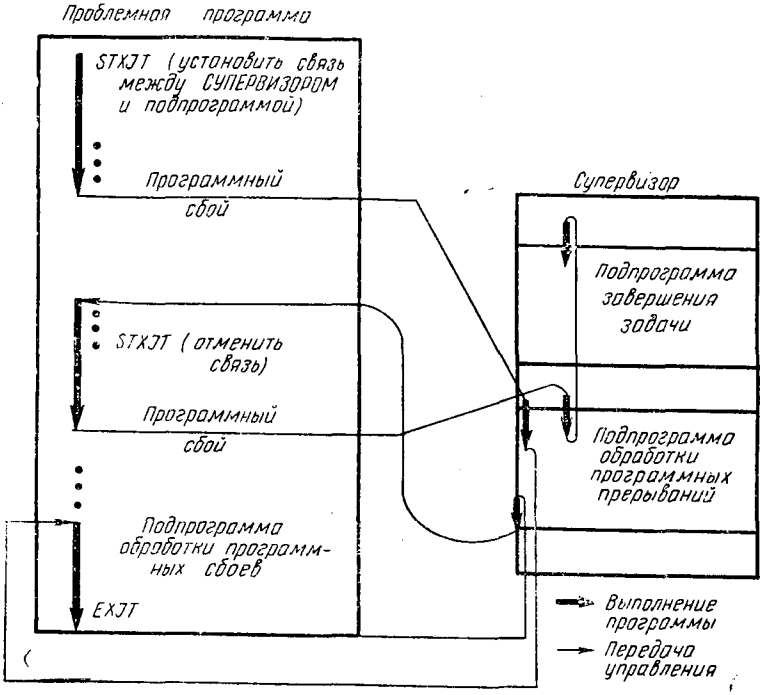


Рис. 11.2. Схема обработки прерываний по программным сбоям

По прерываниям, вызванным схемами контроля машины, СУПЕРВИЗОР переводит машину в состояние ожидания, все прерывания запрещаются. При этом в байты основной памяти с адресами нуль и один записываются специальные коды. Эти коды означают, что прерывание вызвано либо машинным сбоем, либо сбоем управления канала, или сбоем интерфейса между каналом и процессором. После возникновения прерываний от схем контроля машины продолжение работы системы невозможно. Для восстановления операционной системы необходима процедура начальной загрузки.

Сигналы ввода-вывода, вызывающие прерывания, появляются при освобождении канала, при освобождении устройства управления, при освобождении устройства ввода-вывода, при нажатии клавиши ВНИМАНИЕ на пульте машины.

Причинами внешних прерываний являются внешние сигналы, поступающие по линиям связи и предназначенные, в основном, для обслуживания процессов в реальном времени, сигналы от таймера, сигналы от кнопки ПРЕРЫВАНИЕ на пульте оператора.

При возникновении причины прерывания вычислительная машина автоматически выполняет следующие действия:

1. Запоминает состояние прерванной программы в определенных ячейках постоянно фиксируемой области памяти. Состояние, записанное в эти ячейки, называется ССП. Для каждого класса прерываний отведены свои ячейки для запоминания ССП.

2. Передает управление программам *Р*3 СУПЕРВИЗОРа. Информация, необходимая для пуска программ *Р*3 СУПЕРВИЗОРа, хранится также в постоянно фиксированной области основной памяти в ячейках, называемых НСП. Аналогично ССП для каждого класса прерываний выделены свои ячейки для хранения НСП.

Таким образом, механизм прерывания заключается в замене ССП на НСП. НСП определяют точки входа в СУПЕРВИЗОР по каждому классу причин прерываний.

Получив управление в результате прерывания, СУПЕРВИЗОР запоминает полное состояние прерванной программы, определяемое текущим состоянием программы, содержимым общих регистров, регистров с плавающей запятой и другой информацией. Это необходимо для обеспечения возможности продолжить выполнение программы после обработки прерывания. Состояние программы запоминается в области памяти прерванной программы, которая называется ОБЛАСТЬЮ ПАМЯТИ СУПЕРВИЗОРА.

После запоминания состояния программы СУПЕРВИЗОР выполняет обработку прерывания, которая заключается в уточнении причины прерывания и выполнении определенных действий, связанных с этой причиной прерывания.

11.3. Организация обработки прерываний

Все действия, выполняемые СУПЕРВИЗОРом, можно разделить на две группы: 1) действия, которые должны быть выполнены немедленно, или за время до возникновения следующего прерывания; 2) действия, которые могут быть прерваны или приостановлены.

В общем случае первая группа действий выполняется программами *Р*3, вторая группа действий — программами *Р*2. Однако существуют факторы, определяющие, в каком состоянии процессора должна быть выполнена соответствующая группа действий. К таким факторам относятся частота использования и длительность действия. Назовем отрезок времени, в течение которого производится вход в программу, выполнение ее, выбор следующей программы и передача управления выбранной программе, временем действия. Если в течение времени действия не может произойти прерывание, назовем его непрерываемым.

Программы, время действия которых является непрерываемым, могут выполняться в состоянии *Р*2. Например, некоторые программы, обслуживающие операции ввода-вывода, выполняются в состоянии процессора *Р*2. Программы распознавания и обработки,

требующие модификации некоторых управляющих таблиц СУПЕРВИЗОРа, должны выполняться в состоянии *Р3* либо иметь непрерываемое время действия.

На рис. 11.3 приведена диаграмма обработки прерываний. Она организована следующим образом.

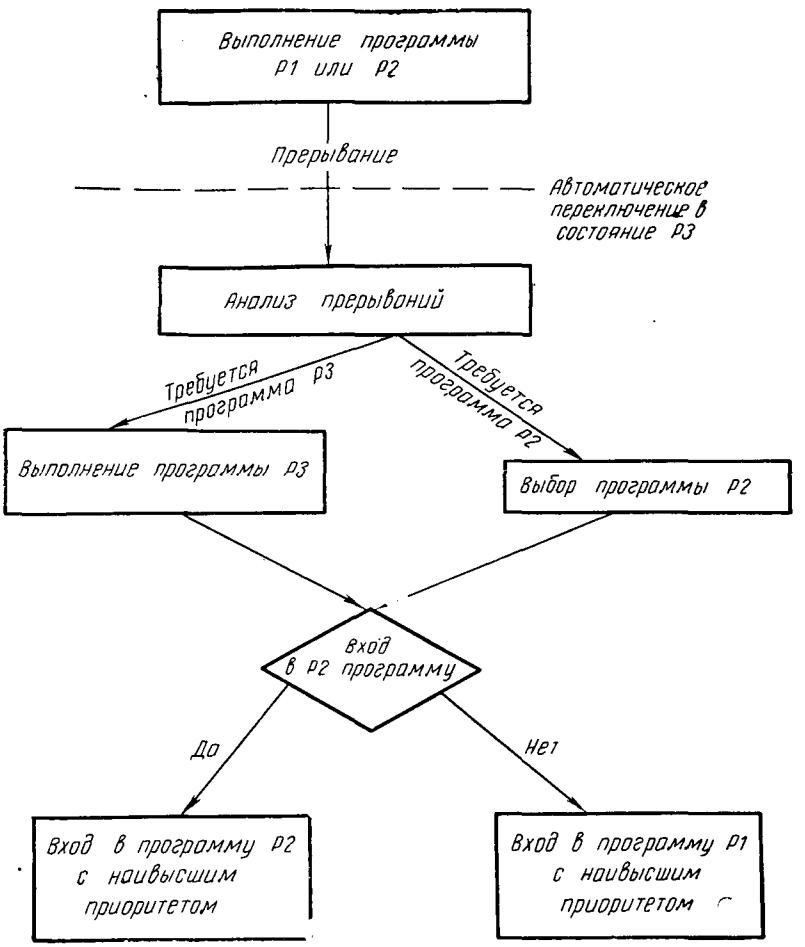


Рис. 11.3. Схема обработки прерываний

После возникновения сигнала прерывания вычислительная машина передает управление программам *Р3*. Программы *Р3* анализируют прерывание и вызывают к действию требуемую программу обработки прерываний. Если требуемая программа обработки принадлежит программам *Р2*, то сначала производится выбор нужной программы. Поиск нужной программы *Р2* осуществляется программами *Р3* по ТАБЛИЦЕ ЗАДАЧ, которая является одной из

управляющих таблиц СУПЕРВИЗОРа. Если найденная программа *Р2* находится в основной памяти в состоянии готовности (т. е. не ждет выполнения какого-либо события), то она получает управление. Если найденная программа *Р2* находится в состоянии ожидания, то управление передается программе *Р1* с наивысшим приоритетом.

Программы *Р3* обладают наивысшим приоритетом по сравнению с программами *Р2* и *Р1*. Их выполнение всегда предшествует выполнению программ *Р2* и *Р1*. Программы *Р3* заканчивают свое выполнение передачей управления программе ВЫБОР, которая определяет, какой программе *Р2* или *Р1* передать управление.

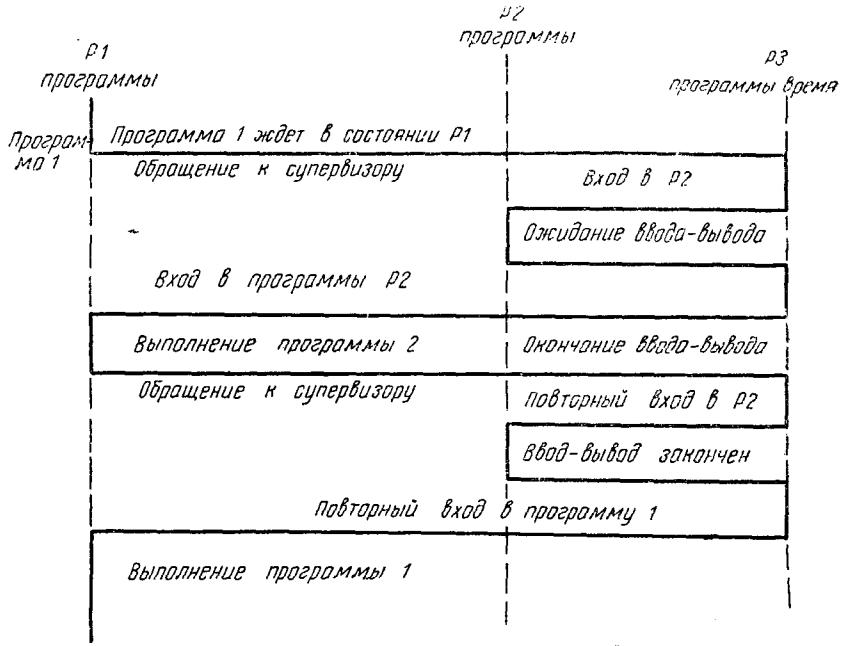


Рис. 11.4. Пример обработки прерываний

Таким образом, существует единственный метод входа в программы *Р3* и единственный метод выхода. Вход в программы *Р3* осуществляется автоматически при возникновении причины прерывания путем выбора нового слова состояния программы из постоянно фиксируемой области памяти. Выход из программ *Р3* осуществляется передачей управления программе ВЫБОР или переходом в состояние процессора *Р4*, если появился сигнал прерывания от схем контроля машины.

Все программы *Р3* являются непрерываемыми, т. е. их действия заканчиваются прежде чем может возникнуть причина прерывания (за исключением сигналов прерываний от схем контроля машины). Все программы *Р2* и *Р1* являются прерываемыми, что создаст возможность параллельно выполнять обработку нескольких причин

прерываний. Программы $P2$ и $P1$ получают управление согласно их приоритету. Управление получает программа с наивысшим приоритетом. Все программы $P2$ имеют приоритет выше любой программы Pi.

На рис. 11.4 приведен простой пример обработки прерывания для выполнения операций ввода-вывода для программы $P1$.

Пусть в состоянии $P1$ выполняется программа ПРОГ1. В некоторый момент времени ей требуется выполнить операцию ввода-вывода. ПРОГ1 переходит в состояние ожидания. Происходит прерывание в состояние $P3$, программы которого анализируют причину прерывания и вызывают соответствующую программу $P2$. Программа $P2$ обращается к СУПЕРВИЗОРу с запросом на инициирование операции ввода-вывода. Управление снова переходит к программам $P3$, которые инициируют операцию ввода-вывода. Выполнение операций ввода-вывода совмещено с выполнением программ, поэтому процессор освобождается. Так как программа $P2$ должна ждать окончания операции ввода-вывода, управление передается в состояние процессора $P1$. Программа ПРОГ1 находится также в состоянии ожидания. Поэтому управление передается программе ПРОГ2, имеющей наивысший приоритет. ПРОГ2 выполняется до тех пор, пока не закончится операция ввода-вывода для ПРОГ1, которая имеет более высокий приоритет.

По окончании операции ввода-вывода происходит прерывание в состояние $P3$. Программы $P3$ анализируют прерывание и снова передают управление программе $P2$, которая вновь вызывает СУПЕРВИЗОР для выполнения действий по окончании ввода-вывода. Снова происходит прерывание в состояние $P3$, программы которого определяют, что $P2$ свободна и ПРОГ1 может продолжить свою работу. Управление передается в состояние $P1$ программе ПРОГ1.

Мы рассмотрели некоторые основные положения организации обработки прерываний. Все сказанное можно суммировать в табл. 11.1.

Таблица 11.1

Состояние процессора	Статус	Назначение	Метод входа
$P1$	полностью прерываемое	обрабатывающие программы	по командам управления программами
$P2$	прерываемое	программы СУПЕРВИЗОРа, реализующие его функции	по командам управления программами
$P3$	непрерываемое, кроме прерываний от схем контроля машины	программы распознавания прерываний и управляющие таблицы	обычные прерывания
$P4$	непрерываемое	программы обработки прерываний от схем контроля машины	прерывания от схем контроля машины

ВОПРОСЫ К ГЛАВЕ

1. Назначение СУПЕРВИЗОРа.
2. Что составляет ядро СУПЕРВИЗОРа?
3. Понятие резидентных и транзитных программ.
4. Состав СУПЕРВИЗОРа.
5. Назначение программ *Р3*.
6. Назначение программ *Р2*.
7. Какова структура области управляющей программы?
8. Особенность программных прерываний.
9. Реакция СУПЕРВИЗОРа на прерывания от схем контроля машины.
10. Действия машины при возникновении прерываний.
11. Статус программ *Р1*, *Р2*, *Р3*, *Р4*.

Глава 12

СУПЕРВИЗОР ЗАДАЧ

12.1. Управляющие таблицы

К СУПЕРВИЗОРу задач относится совокупность управляющих таблиц и программ *Р3*, которые создают задачи и управляют их выполнением. Задачи, формируемые в системе, могут быть двух типов: системные и проблемные. Системные задачи служат для обеспечения тех функций СУПЕРВИЗОРа, которые были определены программами *Р3*. Им подчинены программы *Р2*. Программа, которая начинает выполняться при получении задачей управления, называется подчиненной данной задаче.

Проблемные задачи формируются при инициировании шага задания или программным путем посредством специальной макрокоманды ATTACH. Проблемным задачам подчинены программы *Р1*.

Задача может обратиться к другой задаче с требованием выполнить для нее некоторые функции. В том случае задача, которая осуществляет вызов другой задачи, называется вызывающей. Задача, к которой обратилась вызывающая задача, называется вызываемой.

Вызывающая задача сама может быть вызвана какой-либо еще задачей, для которой она является вызываемой задачей. Аналогично вызываемая задача может быть вызывающей по отношению к следующей задаче. Таким образом, получаем цепочку связей задач, которая находит отражение в ряде управляющих таблиц и которая определяет последовательность вызова одних задач другими. Задача начинает выполняться только в том случае, если она вызвана какой-либо другой задачей. Такая задача называется активной. Задача называется занятой по отношению к другой задаче, если она активна для одной задачи и к ней обратилась другая задача. Последняя вызывающая задача называется ожидающей. Для вызываемой задачи организуется счетчик числа ожидаемых ею задач.

Управление задачами заключается в построении цепочки связей задач, которая отражается в управляющих таблицах и в анализе этих таблиц для принятия решения, какой задаче следует передать управление. При этом не делается различия между системными и проблемными задачами. Управление передается задаче с наивысшим приоритетом. Так как системные задачи предназначены для обслуживания требований на ресурсы системы, то они имеют более высокий приоритет, чем проблемные задачи.

Метод управляющих таблиц возник на ранних стадиях развития автоматизации программирования. Он нашел широкое применение при проектировании управляющих и диспетчерских программ, трансляторов, стандартных процедур для оптимизации их работы. Этот метод положен в основу проектирования всех компонентов операционной системы. При этом основные управляющие таблицы входят в состав СУПЕРВИЗОРа и обрабатываются в состоянии процессора *Р3*. Мы рассмотрим важнейшие управляющие таблицы, раскрывающие принципы работы СУПЕРВИЗОРа и других компонентов операционной системы.

ТАБЛИЦА АДРЕСОВ SYSAT — одна из основных управляющих таблиц СУПЕРВИЗОРа. Она предназначена для организации быстрого доступа к любой управляющей таблице и содержит адреса всех таблиц, используемых СУПЕРВИЗОРом и другими компонентами операционной системы, а также некоторую информацию, характеризующую текущее состояние этих таблиц. Структура и размер ТАБЛИЦЫ АДРЕСОВ зависят от мощности операционной системы (наличия тех возможностей, которые предоставляются программисту), числа используемых СУПЕРВИЗОРом таблиц. В ней хранятся, например, адреса таблицы информации об устройствах, адрес таблицы задач, списка распределенной памяти, очереди каналов, указателя распределения памяти, таблицы приоритетов, списка сообщений оператора, начальный адрес области связи, флаги состояния таблиц и другая информация. В ней также находится число программ *Р2* в данной конкретной конфигурации СУПЕРВИЗОРа, полученной при генерации операционной системы, и адрес информации о задаче, которой СУПЕРВИЗОР передал управление.

ТАБЛИЦА АДРЕСОВ создает предпосылки для независимости программ обработки, входящих в СУПЕРВИЗОР, а также предоставляет возможность эффективно использовать модульность СУПЕРВИЗОРа и возможность добавления новых программ. Адрес этой таблицы хранится в постоянно фиксируемой области основной памяти в байтах 84—85 (табл. 2.1).

Вход в программы *Р3* осуществляется машиной автоматически в зависимости от класса прерываний. Программы *Р3* анализируют прерывание и передают управление либо соответствующей программе обработки прерывания *Р2*, либо программе *Р1*. Выбор нужной программы *Р2* или *Р1* осуществляется с помощью ТАБЛИЦЫ ЗАДАЧ, важнейшей из управляющих таблиц СУПЕРВИЗОРа.

Для управления задачей формируется блок управления задачей, который содержит кодированную информацию об этой задаче. Совокупность блоков управления задачами образует ТАБЛИЦУ ЗАДАЧ. Как только в ТАБЛИЦЕ ЗАДАЧ отводится участок для размещения блока управления задачей, происходит инициирование задачи. Для проблемной задачи инициирование означает окончательное формирование задачи из шага задания, которое выполняется УПРАВЛЕНИЕМ ЗАДАНИЯМИ. Для системных задач инициирование означает, что потребовалось выполнение той или иной функции СУПЕРВИЗОРа.

Кодированная информация, содержащаяся в блоке управления задачей, предназначена для пуска программы, подчиненной данной задаче, для восстановления программы после прерывания, для правильного продолжения и для анализа состояния задачи.

ТАБЛИЦА ЗАДАЧ имеет фиксированную структуру с точки зрения размещения блоков управления системными и проблемными задачами. Местоположение блока управления задачей в ТАБЛИЦЕ ЗАДАЧ называется точкой входа задачи в ТАБЛИЦУ. Порядковый номер точки входа играет важную роль. Он идентифицирует задачу и является ее адресом в ТАБЛИЦЕ ЗАДАЧ. На его основе определяется приоритет задачи. В дальнейшем будем называть его номером задачи или номером точки входа задачи.

В более ранних проектах операционных систем номера точек входов системных задач фиксировались независимо от того, является ли подчиненная им программа резидентной или нет. Это затрудняло модификацию программ *Р2* и приводило к реорганизации ТАБЛИЦЫ ЗАДАЧ при добавлении новых программ *Р2*. Поэтому в современных операционных системах используется более гибкий аппарат выделения участка системным задачам в ТАБЛИЦЕ ЗАДАЧ.

Системным задачам, которым подчинены резидентные программы *Р2*, выделены фиксированные номера точек входов в ТАБЛИЦУ ЗАДАЧ. Эти задачи образованы заранее при генерации системы и поэтому находятся в состоянии готовности. Системные задачи, которым подчинены транзиты *Р2*, формируются в момент загрузки транзитов в основную память. Им также выделены фиксированные номера точек входа. Однако число точек входов значительно меньше, чем число самих транзитных программ, т. е. различные транзиты *Р2* могут подчиняться одной системной задаче. Естественно, что такие транзиты не могут одновременно находиться в основной памяти. Механизм определения номера системной задачи, которой подчинен тот или иной транзит *Р2*, более подробно будет рассмотрен при изучении ТАБЛИЦЫ НОМЕРОВ СИСТЕМНЫХ ЗАДАЧ.

Таким образом, порядковые номера точек входов всех системных задач всегда фиксированы. Это значит, что число системных задач также фиксировано. Однако системным задачам, которым подчинены транзиты *Р2*, в разные моменты функционирования системы могут подчиняться разные транзиты *Р2*.

Понятие номера точки входа в ТАБЛИЦУ ЗАДАЧ очень важно. На основе номера точки входа определяется приоритет задачи. Задачам отводятся участки в ТАБЛИЦЕ ЗАДАЧ согласно их приоритету. Задачи с высшим приоритетом располагаются в верхней части таблицы и имеют наименьшие номера точек входов. Задачи с наименьшим приоритетом располагаются в нижней части таблицы и имеют наибольшие номера точек входов. Структура ТАБЛИЦЫ ЗАДАЧ приведена на рис. 12.1.

Номера точек входа системных задач предшествуют номерам точек входа проблемных задач. Следовательно, приоритет системных задач всегда выше приоритета проблемных задач.

В отличие от приоритета системных задач приоритет проблемных задач не зависит от номера занимаемого ими участка и может быть изменен во время выполнения по отношению к другим проблемным задачам. При этом изменение приоритета проблемной задачи не означает физического перемещения точки входа внутри ТАБЛИЦЫ ЗАДАЧ. Фиксация приоритета проблемных задач с соответствующим перемещением осуществляется **с помощью таблицы приоритетов проблемных задач.**

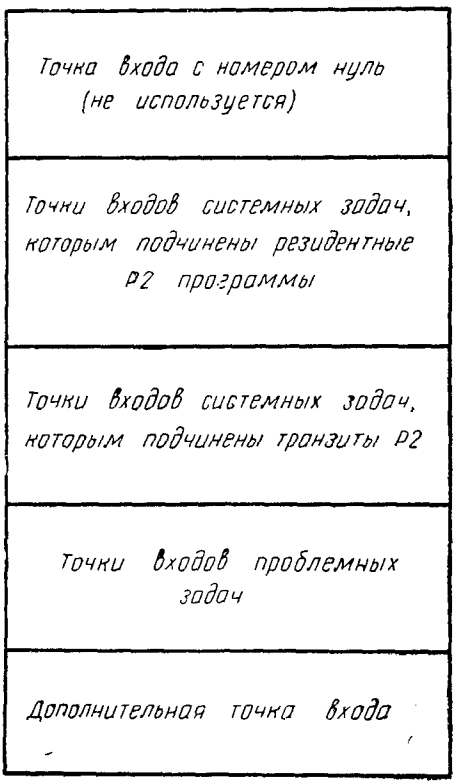


Рис. 12.1. Структура таблицы задач

Таким образом, приоритет резидентных программ *Р2* всегда фиксирован, так как фиксированы номера их блоков управления в ТАБЛИЦЕ ЗАДАЧ. Приоритет транзитов *Р2* фиксирован в текущий момент времени, когда во время их загрузки в основную память формируются задачи, которым отводятся участки в ТАБЛИЦЕ ЗАДАЧ. Это означает, что приоритет транзитов *Р2* между собой не закреплен, но всегда ниже приоритета резидентных программ *Р2*. Приоритет программ *Р1* ниже приоритета всех программ *Р2* и может быть изменен во время выполнения.

Подобный способ формирования ТАБЛИЦЫ ЗАДАЧ определяет простой механизм выбора программы, которой СУПЕРВИЗОР

должен передать управление в текущий момент времени. Так как управление передается всегда программе с наивысшим приоритетом, СУПЕРВИЗОРу достаточно просмотреть ТАБЛИЦУ ЗАДАЧ сверху вниз, найти задачу, готовую к выполнению, и передать управление программе, подчиненной данной задаче.

Число участков, отводимых в ТАБЛИЦЕ ЗАДАЧ системным и проблемным задачам, определяется во время генерации СУПЕРВИЗОРа и хранится вместе с адресом ТАБЛИЦЫ ЗАДАЧ в ТАБЛИЦЕ АДРЕСОВ. Блоки управления проблемными задачами формируются программой ГЛАВНЫЙ ПЛАНИРОВЩИК, являющейся составной частью УПРАВЛЕНИЯ ЗАДАНИЯМИ.

Блок управления задачей занимает в ТАБЛИЦЕ ЗАДАЧ участок длиной в шестнадцать байтов. Структура блока управления задачей приведена в табл. 12.1.

Таблица 12.1

Байты	Содержание	Идентификатор поля
0—3	Содержимое Р-счетчика	А
4—6	Условие ожидания	Б
7	Номер точки входа вызываемой задачи	В
8	Номер точки входа задачи, которую ожидает данная задача	Г
9—11	Начальный адрес программы, подчиненной данной задаче	Д
12	Номер точки входа вызывающей задачи	Е
13	Число задач, ожидающих данную	Ж
14	Номер пульта оператора	З
15	Ключ защиты памяти	И

Рассмотрим назначение отдельных полей блока управления задачей с кратким пояснением, в какой момент происходит формирование этих полей.

Поле А предназначено для запоминания значения Р-счетчика того состояния процессора, в котором была прервана выполняемая последней программа. Значение Р-счетчика запоминается в блоке управления той задачи, подчиненная программа которой была прервана в результате возникшей причины прерывания. После обработки прерывания управление необходимо вернуть прерванной программе для продолжения выполнения. Для восстановления содержимое поля А будет пересылаться в Р-счетчик того состояния процессора, в котором выполнялась прерванная программа.

Если прерывание произошло по команде обращения к СУПЕРВИЗОРу, значение Р-счетчика будет указано как параметр к команде и может быть модифицировано программами *Р*3 и *Р*2.

Поле Д хранит адрес программы *Р*2, которая подчинена данной задаче. Когда данная задача получает управление, значение поля Д

пересылается в поле А, чтобы не нарушить общий порядок выполнения задач. Поле Д используется для связи только для системных задач, а для проблемных задач всегда равно нулю. Поле Д формируется в момент загрузки программ *Р2* в основную память и является пусковым адресом этой программы.

Поле Б предназначено для хранения условия ожидания. Если задача ждет выполнения какого-либо события, например окончания операции ввода-вывода или загрузки какой-либо транзитной программы, в поле Б записывается код условия ожидания. Если задача не ждет, поле Б очищается нулями.

Поле В предназначено для хранения номера блока управления вызванной задачи. Когда какой-либо задаче (проблемной или системной) требуется выполнение одной из функций СУПЕРВИЗОРа, подчиненная ей программа вырабатывает команду обращения к СУПЕРВИЗОРу с номером требуемой программы *Р2*. Посредством ТАБЛИЦЫ ПРИОРИТЕТОВ СИСТЕМНЫХ ЗАДАЧ и ТАБЛИЦЫ ЗАДАЧ определяется номер блока управления соответствующей задачи, которой подчинена требуемая программа *Р2*. Найденный номер блока управления запоминается в поле В вызывающей задачи, если вызванная задача не занята. Задача считается незанятой, если поле Е ее блока управления равно нулю.

Если вызванная системная задача занята (ее поле Е отлично от нуля), то номер ее точки входа запоминается не в поле В, а в поле Г вызывающей задачи. Одновременно в поле Ж вызванной задачи добавляется единица.

Поле Е предназначено для хранения номера точки входа вызывающей задачи.

Когда одна задача вызывает другую, то в поле В вызывающей задачи ставится номер точки входа вызываемой задачи, в поле Е вызываемой задачи ставится номер точки входа вызывающей задачи. Заполнение поля Е означает, что вызываемая задача активна. Если теперь другая вызывающая задача обратится к активной задаче, то у новой вызывающей задачи заполняется поле Г (вместо поля В), а в активной вызываемой задаче в поле Ж добабится единица.

Поле Ж содержит число задач, ожидающих данную, т. е. число задач, в поле Г которых указан номер блока управления данной задачи. Когда программа, подчиненная данной задаче, окончит свои действия, она с помощью команды обращения к СУПЕРВИЗОРу SVC 32 вызовет программу ВОЗВРАТ. Программа ВОЗВРАТ проверяет, равно ли поле Ж задачи, окончившей свои действия, нулю. Если значение поля Ж не равно нулю, оно уменьшается на единицу, затем ищется первая задача, в поле Г которой указан номер блока управления задачи, окончившей выполнение. В найденной задаче содержимое поля Г пересылается в поле В, поле Г очищается и поле В пересылается в поле Е вызываемой задачи обычным путем. Поле Д вызываемой задачи пересылается в поле А и управ-

ление передается программе ВЫБОР (действия программ ВОЗВРАТ и ВЫБОР будут рассмотрены позднее).

Поле З предназначено для указания номера пульта, на который в случае необходимости надо выдать сообщение.

Если машина имеет несколько пультов оператора, они распределяются между проблемными задачами в соответствии с теми потоками, в которых эти задачи выполняются. Присвоение номера пульта проблемной задаче осуществляет программа ГЛАВНЫЙ ПЛАНИРОВЩИК во время инициирования проблемной задачи. Когда одна задача вызывает другую, поле З копируется из блока управления вызывающей задачи в блок управления вызываемой задачи.

Поле И хранит ключ защиты памяти, выделенной данной задаче. Для всех системных задач ключ защиты памяти равен нулю, кроме задач, которым подчинены программы *Р2* обнаружения ошибок, ключ защиты памяти которых равен пятнадцати. Ключ защиты памяти проблемных задач устанавливается программой ГЛАВНЫЙ ПЛАНИРОВЩИК во время инициирования задачи. Задача считается инициированной, если ей отведен участок в ТАБЛИЦЕ ЗАДАЧ для размещения блока управления.

Если задача не имеет ключа защиты, то поле И заполняется всеми единицами.

Дополнительная точка входа (см. рис. 12.1) является последним участком в ТАБЛИЦЕ ЗАДАЧ. Она никогда не вызывается проблемными задачами. Назначение дополнительной точки входа — организовать цепочку задач в тех ситуациях, когда надо активизировать задачу, которая в настоящий момент не вызывается никакой другой задачей. Например, программа, которая вызывается каждые десять секунд для обработки прерывания от часов, является одной из программ *Р2* распознавания ошибок.

Для обращения к такой программе в любое время необходимо, чтобы соответствующая ей задача была активизирована, т. е. в ее блоке управления в ТАБЛИЦЕ ЗАДАЧ в поле Е стоял номер вызывающей задачи. Такой вызывающей задачей может стать фиктивная задача, соответствующая дополнительной точке входа. Фиктивная задача, которой соответствует дополнительная точка входа, вызывает данную задачу обработки прерывания от часов. Вызов осуществляется обычным образом. Если вызванная таким образом задача уже занята, т. е. поле Е в ее блоке управления отлично от нуля, то вызов от фиктивной задачи игнорируется (номер дополнительного участка не записывается в поле Г). Если задача вызвана фиктивной задачей, то поля Б, В, Г в дополнительной точке входа никогда не заполняются.

Рассмотренная нами ТАБЛИЦА ЗАДАЧ используется в управляющих программах, реализующих мультипрограммирование с фиксированным числом задач. С ее помощью организуется инициирование задачи, выделение задаче ресурсов, формирование очередей задач, их синхронизация и управление выполнением задач. Доступ

к этой таблице имеют лишь немногие программы *Р3*, которые мы изучим в этой главе. Без понимания этой таблицы невозможно понимание функционирования системы в целом. Чтобы уточнить назначение отдельных полей блока управления задачей, рассмотрим пример работы с таблицей.

Пусть в текущий момент времени ТАБЛИЦА ЗАДАЧ имеет вид табл. 12.2, в которой каждая строка соответствует одному блоку управления задачей. В первой графе указывается порядковый номер блока управления задачей. Для простоты изложения будем считать его номером задачи. Проанализируем состояние таблицы.

Начнем просмотр таблицы сверху вниз. Задачи, которым соответствуют первая и вторая точки входа, находятся в состоянии ожидания, так как в поле Б записаны коды условия ожидания. В поле Е третьей задачи стоит номер 72 вызывающей задачи. Следовательно, в поле В вызывающей задачи 72 должен стоять номер вызываемой программы — три. Третья задача не может выполняться, так как она, в свою очередь, вызвала задачу с номером четыре, о чем говорит поле В. В вызываемой четвертой задаче заполнено поле Е, где указан номер вызывающей задачи — три. Задача четыре также не может выполняться, так как она ожидает окончания какого-либо события. В поле Б записан код условия ожидания, например окончание операции ввода-вывода. В поле Ж программы три находится число ожидающих ее программ. Это значит, что в поле Г какой-либо вызывающей задачи должен стоять номер три вызываемой задачи. Такой задачей является задача 73. Программа пять вызвана из дополнительной точки входа и ее ожидает задача восемь. Так как все задачи с более высоким приоритетом, чем пять, находятся в состоянии ожидания, то задача пять является той задачей, которая может получить управление.

Допустим, что в это время окончилась операция ввода-вывода, которую ожидала задача четыре. Приоритет четвертой задачи выше приоритета программы пять, следовательно, управление должно быть передано задаче четыре, которая начинает выполнять свои действия по требованию третьей задачи. В некоторый момент времени задача четыре закончила свою работу для задачи три, которая освобождается для выполнения по вызову задачи 72. Поля Б и Е задачи четыре и поле В задачи три очищаются. Управление получает задача три как задача с наивысшим приоритетом. По окончании работы задачи три по вызову от задачи 72 поле Е задачи три и поле В задачи 72 очищаются. Из поля Ж задачи три вычитается единица. Поле Г ожидающей задачи 73 пересылается в поле В ее блока управления, а номер ее пересылается в поле Е задачи три. Теперь ТАБЛИЦА ЗАДАЧ будет иметь вид, приведенный в табл. 12.3.

СУПЕРВИЗОР осуществляет выбор нужной программы обработки прерываний путем формирования задачи и размещения ее блока управления в ТАБЛИЦЕ ЗАДАЧ. Все обстояло бы очень просто, если бы каждой программе *Р2* соответствовала задача со

своим номером точки входа. Однако число программ *Р2*, транзитных и резидентных, превышает число участков, отводимых системным задачам, которым они подчинены. При этом резидентным программам *Р2* соответствуют системные задачи с фиксированными номерами точек входов в ТАБЛИЦУ ЗАДАЧ. Другими словами, системные задачи, которым подчиняются резидентные программы *Р2*, имеют фиксированные номера и заранее определенные адреса местоположения их блоков управления в ТАБЛИЦЕ ЗАДАЧ.

Таблица 12.2

№ п/п	А	Б	В	Г	Д	Е	Ж	З	И	Примечание
0										не используется
1		У 01								ждут
2		У 02								
3			04			72	01			
4		У 03				03				
5						75	01			
⋮										
8				05		74	01			
9										
10				08		75			OF	
⋮										
72			03						01	
73				03					02	
74			08						03	
75										дополнительная точка входа

Для каждой транзитной программы *Р2* формируется системная задача со строго определенным номером. Так как номеров задач меньше, чем транзитных программ, различным транзитным про-

граммам будут соответствовать системные задачи с одним и тем же номером. Определение номера системной задачи для подчиненных транзитных программ осуществляется посредством ТАБЛИЦЫ НОМЕРОВ СИСТЕМНЫХ ЗАДАЧ.

Таблица 12.3

№ п/п	А	Б	В	Г	Д	Е	Ж	З	И	Примечание
0										
1										
2										
3						73				
4										
5						75	01			
⋮										
8				05		74	01			
9										
10				08		75			OF	
⋮										
72										
73			03						02	
74		08							03	
75										дополнительная точка входа

Каждая программа *Р2*, резидентная или транзитная, имеет собственный номер. Казалось бы, что резидентные программы *Р2* не имеет смысла включать в ТАБЛИЦУ НОМЕРОВ СИСТЕМНЫХ ЗАДАЧ, так как им соответствуют фиксированные номера задач в ТАБЛИЦЕ ЗАДАЧ и можно номера резидентных программ сделать равными номерам их задач. Однако некоторые транзитные программы *Р2* могут быть по желанию пользователя определены как резидентные в момент генерации СУПЕРВИЗОРа, а часть рези-

дентных программ вообще не включаться в СУПЕРВИЗОР или определяться как транзитные. Это значит, что каждая конфигурация операционной системы имеет различные наборы резидентных и транзитных программ *P2* из общего числа всех программ *P2*. Поэтому ТАБЛИЦА НОМЕРОВ СИСТЕМНЫХ ЗАДАЧ содержит номера всех программ *P2*.

Основное назначение ТАБЛИЦЫ НОМЕРОВ СИСТЕМНЫХ ЗАДАЧ — связать номера программ *P2* с номерами соответствующих им задач для размещения блоков управления в ТАБЛИЦЕ ЗАДАЧ. Структура ТАБЛИЦЫ приведена на рис. 12.2.

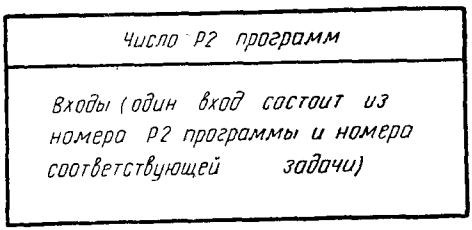


Рис. 12.2. Структура таблицы номеров системных задач

ТАБЛИЦА НОМЕРОВ СИСТЕМНЫХ ЗАДАЧ содержит столько входов, сколько существует программ *P2* в конкретной конфигурации операционной системы. Число программ *P2* задается в двоичной системе счисления и занимает два байта. Каждый вход также состоит из двух байтов. В первом байте указывается номер системной задачи, которой подчинена данная программа. Во втором байте указывается номер программы для транзитных программ P2, константа *X'FF'* для стандартных резидентных программ, константа *X'FE'* для программ, объявленных резидентными во время генерации системы.

Номера точек входов системных задач, которым подчинены транзитные программы *P2*, вычисляются заранее по следующему алгоритму. Номера программ сравниваются с числом 32. Это число выбрано в связи с тем, что одновременно в основной памяти не может находиться больше 32 транзитных программ *P2*. Если номер программы не превышает этого числа, то он не модифицируется и считается равным номеру точки входа в ТАБЛИЦУ ЗАДАЧ. Если номер программы превышает число 32, то производится модификация. Для этого из номера программы вычитают число 32 столько раз, чтобы полученное число не превышало 32. Остаток от вычитания является номером точки входа в ТАБЛИЦУ ЗАДАЧ соответствующей системной задачи. Например, программе с номером 33 соответствует задача с номером один, и блок управления задачей будет размещен в ТАБЛИЦЕ ЗАДАЧ в участке с номером один в области точек входов транзитов *P2* (рис. 12.1).

12.2. Анализ прерываний

Рассмотрим основные программы *P3*, осуществляющие управление задачами. К ним относятся АНАЛИЗ ПРЕРЫВАНИЙ, ВОЗВРАТ, СОГЛАСОВАНИЕ, ВЫБОР.

При возникновении прерывания вычислительная машина запоминает слово состояния прерванной программы в постоянно фиксированной области основной памяти и передает управление программе *Р3* СУПЕРВИЗОРа АНАЛИЗ ПРЕРЫВАНИЙ. Эта программа производит анализ прерываний и определяет программу его обработки, для которой производит инициацию задачи для того, чтобы эта программа смогла получить управление.

Функциональная блок-схема АНАЛИЗА ПРЕРЫВАНИЙ приведена на рис. 12.3.

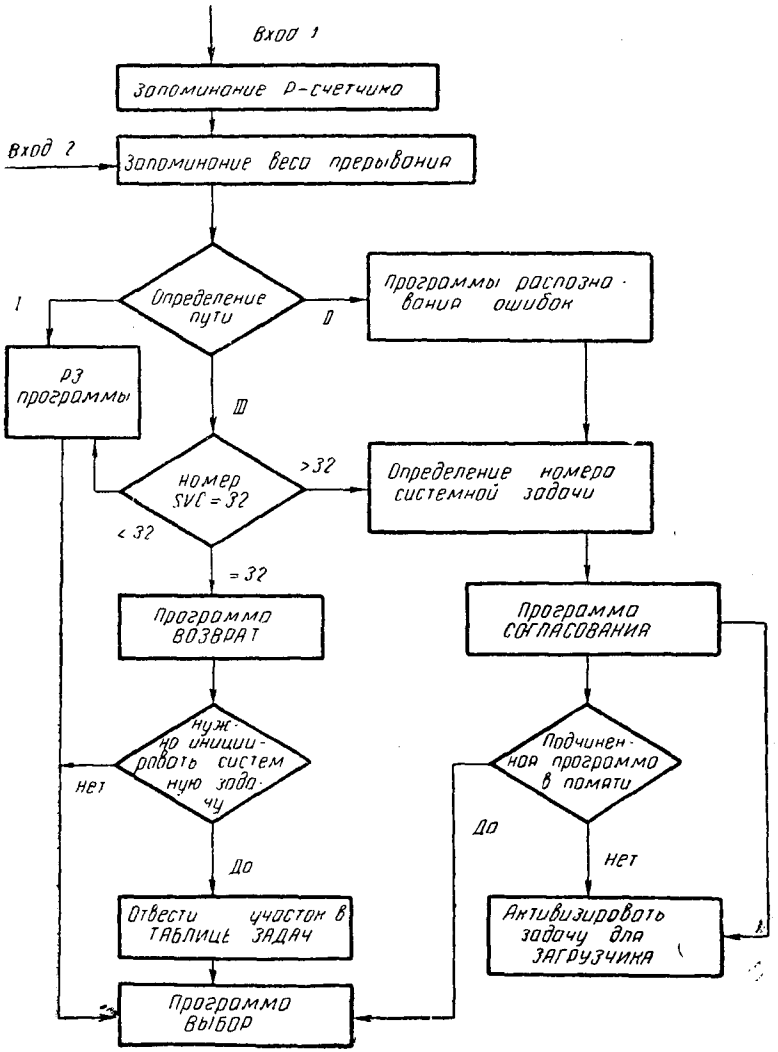


Рис. 12.3. Анализ прерываний

Прерывания от схем контроля машины переводят процессор в состояние *Р4* и программами *Р3* не обрабатываются.

Причина прерывания может возникнуть как при работе программ *Р1* или *Р2*, так и во время работы программ *Р3*. В первом случае необходимо запомнить содержимое Р-счетчика для того, чтобы после обработки прерывания в состоянии процессора *Р3* можно было продолжить выполнение прерванной программы. Во втором случае причина прерывания возникла, когда программа уже выполняется в состоянии процессора *Р3*. Запоминать значение Р-счетчика уже нельзя, так как его первоначальное значение может быть изменено программами *Р3* и после восстановления может привести к путанице. Последнее возможно, когда Р-счетчик из ТАБЛИЦЫ ЗАДАЧ используется как параметр в командах обращения к СУПЕРВИЗОРу. Поэтому имеются два входа в программу *Р3* АНАЛИЗ ПРЕРЫВАНИЙ. На первый вход поступают все прерывания, возникшие при выполнении программ *Р1* или *Р2*, на второй вход поступают все прерывания, возникшие при выполнении программ *Р3*. Управление от первого входа также передается на второй вход.

Второй вход обрабатывает вес прерывания. Каждое прерывание имеет свой вес, который устанавливается аппаратурно в общем регистре 15 состояния процессора *Р3*. Вес запоминается для дальнейшего включения его в ОБЛАСТЬ ПАМЯТИ СУПЕРВИЗОРа (табл. 15.2) прерванной программы и служит индексом для выбора необходимого пути обработки прерывания.

Существуют три пути обработки: прерывания, которые должны быть обработаны за непрерываемое время, обрабатываются программами *Р3*; программные прерывания обрабатываются программами *Р2* распознавания ошибок; прерывания, возникшие по обращению к СУПЕРВИЗОРу, требуют анализа этих обращений.

Первый путь. Прерывания, которые требуют непосредственной обработки за непрерываемое время, должны обслуживаться в состоянии процессора *Р3*. К таким прерываниям относятся прерывания от ввода-вывода и внешние прерывания. Вес прерывания делится на два и является указателем программы *Р3*, которая должна обработать данное прерывание. Совокупность программ, обрабатывающих прерывания от ввода-вывода, образуют физическую систему управления вводом-выводом. После выполнения этих программ *Р3* управление передается программе ВЫБОР, также являющейся программой *Р3*.

Второй путь. Прерывания не требуют обработки за непрерываемое время, поэтому могут обрабатываться в состоянии процессора *Р2*. К ним относятся программные прерывания. Для этих прерываний необходимо определить тип ошибки и номер программы *Р2*, обрабатывающей этот тип ошибки. Затем нужно с помощью ТАБЛИЦЫ НОМЕРОВ СИСТЕМНЫХ ЗАДАЧ определить номер точки входа в ТАБЛИЦУ ЗАДАЧ блока управления задачей, которой подчинена *Р2* программа. Управление передается программе

Р3 СОГЛАСОВАНИЕ, которая инициирует для найденной программы *Р2* задачу и установит для нее цепочку связей в ТАБЛИЦЕ ЗАДАЧ.

Найденная программа распознавания ошибки может либо не находиться в основной памяти, либо находиться. Если найденная программа находится в основной памяти, то установление цепочки в ТАБЛИЦЕ ЗАДАЧ означает активизацию задачи, соответствующей найденной программе *Р2*. Управление получает программа *Р3* ВЫБОР, назначение которой — определить готовую задачу с наивысшим приоритетом и передать ей управление. Если такой задачей является активизированная задача, то она получает управление для выполнения своих функций. Если существуют готовые задачи с более высоким приоритетом, то активизированная задача является готовой к выполнению и ожидающей ресурса — времени процессора. Окончание любой программы заключается в выполнении команды обращения к СУПЕРВИЗОРу.

Если найденная программа *Р2* не находится в основной памяти, то необходимо загрузить ее в основную память. Загрузку программ в основную память осуществляет резидентная программа *Р2*, называемая ЗАГРУЗЧИК. Чтобы ЗАГРУЗЧИК начал выполнять свои функции, необходимо активизировать соответствующую ему задачу. Так как ЗАГРУЗЧИК является резидентной программой, блок управления соответствующей ему задачи занимает фиксированное место в ТАБЛИЦЕ ЗАДАЧ. Активизация ЗАГРУЗЧИКа означает, что в поле Е его блока управления необходимо указать номер вызывающей задачи, т. е. необходимо построить цепочку связей в ТАБЛИЦЕ ЗАДАЧ для системной задачи, соответствующей ЗАГРУЗЧИКу. Такая цепочка связей устанавливается программой *Р3* СОГЛАСОВАНИЕ, которая вновь получает управление, но уже с новыми параметрами.

Программа СОГЛАСОВАНИЕ активизирует задачу, соответствующую ЗАГРУЗЧИКу и передаст управление программе *Р3* ВЫБОР, которая определит активную системную задачу, которой надо передать управление. Если нет других активных задач с более высоким приоритетом, чем задача, соответствующая ЗАГРУЗЧИКу, то ЗАГРУЗЧИК получит управление. По окончании его действий опять произойдет прерывание, и управление вновь получит программа АНАЛИЗ ПРЕРЫВАНИЙ. Так как ЗАГРУЗЧИК завершает свои действия вызовом СУПЕРВИЗОРа посредством выполнения команды обращения к СУПЕРВИЗОРу, то это прерывание будет анализироваться по третьему пути.

Третий путь. Для выяснения пути обработки возникшего прерывания требуется анализ обращений к СУПЕРВИЗОРу. Обращения к СУПЕРВИЗОРу представляют собой макрокоманды, которые определяют отдельные функции СУПЕРВИЗОРа. Эти функции выполняются программами в различных состояниях процессора. Поэтому в зависимости от пути, по какому будет происходить выполнение запрошенной функции СУПЕРВИЗОРа, обращения к СУ-

ПЕРВИЗОРу можно подразделить на три группы: обращения к СУПЕРВИЗОРу с номером SVC меньше 32, обращение к СУПЕРВИЗОРу с номером SVC 32, обращения к СУПЕРВИЗОРу с номерами SVC больше 32.

Обращения к СУПЕРВИЗОРу с номерами SVC меньше 32 определяют действия, выполняемые программами *Р3*. Они должны быть выполнены немедленно, и управление передается программам *Р3*. Указателем программы *Р3* служит номер команды обращения к СУПЕРВИЗОРу.

Отличие программ *Р3* от программ *Р2* заключается в том, что для программ *Р3* не нужно формировать задачу. Они получают управление непосредственно от других программ *Р3* с помощью команд передачи управления, а от программ *Р1* и *Р2* — посредством прерывания этих программ. Программы *Р2* получают управление только путем формирования для них системных задач и активизации этих задач с использованием ТАБЛИЦЫ ЗАДАЧ.

То, что прерывание требует непосредственной обработки программами *Р3*, означает, что проведенного анализа на основании веса прерывания и номера команды обращения к СУПЕРВИЗОРу недостаточно, и требуется более детальный анализ, который осуществляют другие программы *Р3*, кроме АНАЛИЗА ПРЕРЫВАНИЙ.

В основном, обращения к СУПЕРВИЗОРу, требующие выполнения программ *Р3*, определяют действия, связанные с операциями ввода-вывода. Они представляют собой макрокоманды физической системы управления вводом-выводом. Макрокоманды, вызывающие программы *Р3*, делятся на макрокоманды, которые могут использоваться программами *Р1* и *Р2*, и макрокоманды, которые могут использоваться только программами *Р2*. Последние являются привилегированными макрокомандами. Их использование в обрабатывающих программах недопустимо и приводит к программному сбою.

Обращения к СУПЕРВИЗОРу с номерами SVC больше 32 обслуживаются всегда программами *Р2*. Чтобы вызвать требуемую программу обработки, необходимо инициировать соответствующую ей системную задачу. Инициирование задачи заключается в определении номера системной задачи, размещении блока управления задачей в ТАБЛИЦЕ ЗАДАЧ и установлении для данной задачи цепочки связей.

Определение номера системной задачи осуществляется с помощью ТАБЛИЦЫ НОМЕРОВ СИСТЕМНЫХ ЗАДАЧ. Номер системной задачи определяет точку входа в ТАБЛИЦУ ЗАДАЧ. Дальнейшая обработка ведется тем же образом, что и для программ *Р2* распознавания ошибок.

Обращение к СУПЕРВИЗОРу с номером SVC 32 используется программами *Р2* для указания СУПЕРВИЗОРу ЗАДАЧ, что они закончили выполнение своих действий. Это обращение представляет собой макрокоманду RETURN (ВОЗВРАТ). Макрокоманда RETURN используется в программе, подчиненной вызываемой за-

даче, по окончании ее действий, чтобы передать управление программе, подчиненной вызывающей задаче. Программа, соответствующая макрокоманде RETURN, вызывает *Р3* программу ВОЗВРАТ, которая модифицирует ТАБЛИЦУ ЗАДАЧ.

Программа ВОЗВРАТ очищает цепочку связей для вызываемой задачи, подготавливая ТАБЛИЦУ ЗАДАЧ к новому просмотру, который осуществляет программа *Р3* ВЫБОР. Одновременно проверяется, нет ли программ *Р2*, для которых необходимо инициировать задачу. Делается это по следующей причине. Транзитам *Р2* с разными номерами программ может соответствовать один и тот же номер системной задачи. Это может привести к тому, что некоторые программы *Р2* ожидают, пока освободится участок в ТАБЛИЦЕ ЗАДАЧ, в котором можно будет разместить блок управления соответствующей задачей. Для таких программ формируется специальный счетчик, который размещается в ТАБЛИЦЕ АДРЕСОВ. После того как программа ВОЗВРАТ очистила цепочку связей в ТАБЛИЦЕ ЗАДАЧ, может оказаться, что освободился требуемый участок. В этом случае освободившийся участок отводится системной задаче, которой подчинена ожидающая программа, что означает формирование системной задачи. Эта задача находится в состоянии ожидания, так как она ждет загрузки подчиненной программы *Р2*. Активизация этой задачи будет идти обычным путем после установления для нее цепочки связей. Причиной возникновения описанной ситуации может служить, например, переполнение очереди канала или списка сообщений оператору.

Программа АНАЛИЗ ПРЕРЫВАНИЙ оканчивает свои действия передачей управления программе *Р3* ВЫБОР.

12.3. Программа возврат

Программа ВОЗВРАТ — одна из программ *Р3* СУПЕРВИЗОРа, имеющая доступ к ТАБЛИЦЕ ЗАДАЧ и право ее модификации. Основное назначение этой программы — очистить цепочку связей в ТАБЛИЦЕ ЗАДАЧ для задачи, окончившей выполнение, и «освободить» задачи, ожидающие данную. Программа ВОЗВРАТ получает управление от всех системных задач, окончивших свое выполнение. Все программы *Р2* заканчивают свое выполнение командой обращения к СУПЕРВИЗОРу с одним и тем же номером SVC 32. Эта команда вызовет прерывание, и программа АНАЛИЗ ПРЕРЫВАНИЙ передаст управление программе ВОЗВРАТ. Проблемные программы заканчивают свои действия макрокомандой RETURN, макрорасширение которой содержит команду SVC 32.

Функциональная блок-схема программы ВОЗВРАТ приведена на рис. 12.4.

В качестве параметра в команде обращения к СУПЕРВИЗОРу указывается номер системной задачи, закончившей выполнение. По номеру задачи определяется местоположение ее блока управления в ТАБЛИЦЕ ЗАДАЧ.

Определим статус задачи, окончившей выполнение. Если задача выполнялась, то, следовательно, она была активной. Задача является активной, если она вызываемая по отношению к некоторой вызывающей задаче. Это значит, что поля Б, В, Г ее блока управления равны нулю, а в поле Е указан номер вызывающей задачи. Поле Ж будет заполнено в том случае, если эту задачу ожидают какие-либо другие задачи (см. табл. 12.1).

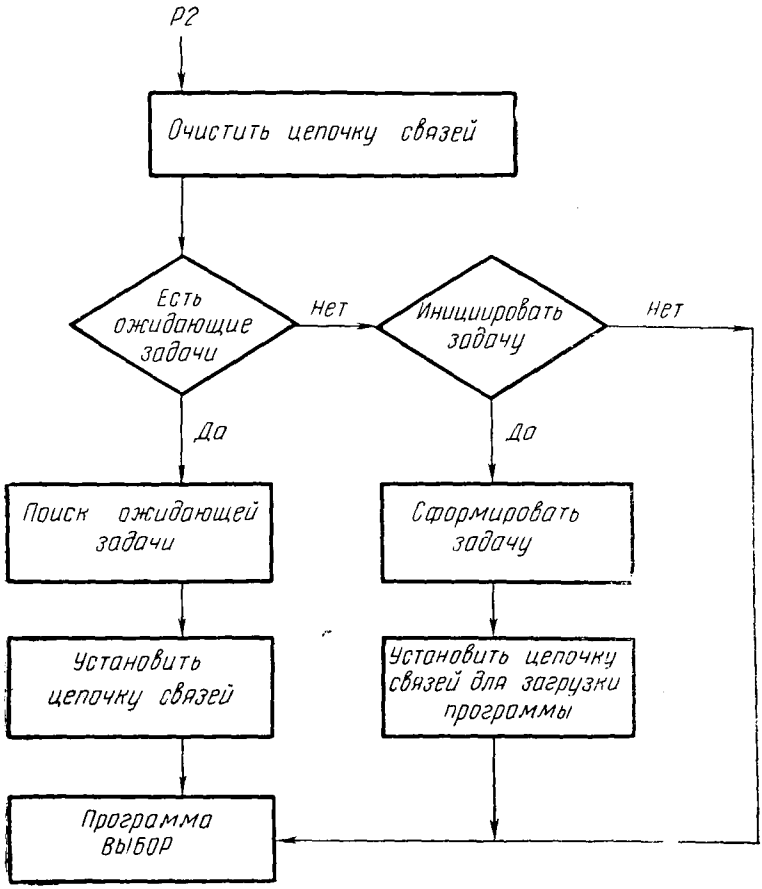


Рис. 12.4. Программа ВОЗВРАТ

Очистка цепочки связей заключается в том, что очищается поле Е вызываемой задачи и поле В вызывающей задачи. Местонахождение блока управления вызывающей задачи определяется по значению поля Е вызываемой задачи.

Мы помним, что номер задачи представляет собой порядковый номер точки входа блока управления в ТАБЛИЦУ ЗАДАЧ. Длина

блока управления задачей фиксирована. Адрес начала ТАБЛИЦЫ ЗАДАЧ $A_т$ хранится в ТАБЛИЦЕ АДРЕСОВ. Следовательно, адрес начала A_n блока управления задачей в ТАБЛИЦЕ ЗАДАЧ определяется по формуле

$$A_n = A_т + 16n.$$

Затем проверяется, равно ли поле Ж вызываемой задачи нулю. Если ожидающих задач нет (поле Ж равно нулю), то задача считается полностью окончившей свои функции, и если подчиненная ей программа является транзитной, то местонахождение этой программы в основной памяти является не обязательным. Это равносильно тому, что участок ТАБЛИЦЫ ЗАДАЧ, занимаемый блоком управления вызываемой задачи, освободился. В этом случае проверяется, нет ли ожидающей программы *Р2*, выполнение которой требуется для обработки какого-либо прерывания. Если такой программы нет, то управление передается программе ВЫБОР. Если такая программа *Р2* есть, то для нее необходимо сформировать системную задачу. Для этого ей отводится участок в ТАБЛИЦЕ ЗАДАЧ для размещения ее блока управления. Номер задачи определяется обычным путем с помощью ТАБЛИЦЫ НОМЕРОВ СИСТЕМНЫХ ЗАДАЧ. Затем для сформированной задачи устанавливается цепочка связей в ТАБЛИЦЕ ЗАДАЧ для загрузки в основную память подчиненной программы, т. е. осуществляется вызов резидентной программы *Р2* ЗАГРУЗЧИК.

Установление цепочки означает, что сформированная задача становится вызывающей и в ее блоке управления в поле В указывается номер вызываемой задачи. В блоке управления вызываемой задачи заполняется поле Е, если задача не активна. Если вызванная задача уже активна, то в блоке управления вызывающей задачи вместо поля В будет заполнено поле Г, а в блоке управления вызываемой задачи в поле Ж будет добавлена единица и никакое другое поле заполняться не будет. После выполнения всех этих действий управление будет передано программе ВЫБОР.

Рассмотрим действия программы ВОЗВРАТ в том случае, если задачу, окончившую выполнение, ожидают другие задачи. В этом случае значение ее поля Ж отлично от нуля. Так как номера ожидающих задач не фиксируются в блоке управления, то их необходимо найти. С этой целью организуется просмотр ТАБЛИЦЫ ЗАДАЧ для поиска задачи, которая ожидает данную. В точке входа ожидающей задачи в поле Г должен быть указан номер точки входа задачи, окончившей выполнение.

Когда ожидающая задача с наивысшим приоритетом найдена, производится установление новой цепочки связей. Для этого значение поля Ж задачи, окончившей выполнение, уменьшается на единицу, в поле Е записывается номер найденной задачи, в поле А пересылается значение поля Д. Задача, окончившая выполнение, становится вновь вызываемой задачей, а найденная задача из ожидающей превращается в вызывающую. В блоке управления вызы-

вающей задачи значение поля Г пересылается в поле В. После этого модификация ТАБЛИЦЫ ЗАДАЧ закончена, и управление передается программе ВЫБОР.

12.4. Программа согласование

Программы СОГЛАСОВАНИЕ и ВОЗВРАТ являются единственными программами *Р3*, которые имеют доступ к ТАБЛИЦЕ ЗАДАЧ и право ее модификации. Основное назначение программы СОГЛАСОВАНИЕ — установить цепочку связей между вызывающей и вызываемой задачами. Номера программ *Р2*, для задач которых устанавливается цепочка связей, указываются в качестве параметров. Первый параметр определяет номер вызывающей задачи, второй параметр — номер программы, подчиненной вызываемой задаче.

Программа СОГЛАСОВАНИЕ имеет два входа. Первый вход предназначен для обслуживания запросов на модификацию ТАБЛИЦЫ ЗАДАЧ, поступающих от системных задач.

Второй вход предназначен для обслуживания программ *Р3*. Все программы *Р3*, которые хотят обратиться к какой-либо программе *Р2*, должны обратиться к программе СОГЛАСОВАНИЕ, чтобы она установила цепочку связей для задачи, которой подчинена эта программа.

Блок-схема программы СОГЛАСОВАНИЕ приведена на рис. 12.5.

Получив управление по первому входу, программа СОГЛАСОВАНИЕ проверяет, закончила ли вызывающая задача свою работу или нет. Если нет, то управление передается на второй вход. Если задача закончила выполнение своих действий, то для нее необходимо очистить связи в ТАБЛИЦЕ ЗАДАЧ. Для этого программа СОГЛАСОВАНИЕ обращается к программе ВОЗВРАТ, как к своей подпрограмме. После этого управление поступает на второй вход. Обращение к программе ВОЗВРАТ как к подпрограмме диктуется тем соображением, что после работы программы ВОЗВРАТ управление должно вернуться программе СОГЛАСОВАНИЕ. Если к программе ВОЗВРАТ обратиться обычным образом посредством команды обращения к СУПЕРВИЗОРу, то управление по окончании ее работы будет передано программе ВЫБОР.

Прежде чем установить цепочку связей между вызывающей и вызываемой задачами, по ТАБЛИЦЕ НОМЕРОВ СИСТЕМНЫХ ЗАДАЧ надо определить номер точки входа вызываемой задачи в ТАБЛИЦУ ЗАДАЧ. Далее определяется, находится ли программа, подчиненная вызываемой задаче, в основной памяти. Для этого анализируется поле Д блока управления вызываемой задачи по ТАБЛИЦЕ ЗАДАЧ (табл. 12.1 и 12.2).

Если значение поля Д равно нулю, то программа не находится в основной памяти, и СОГЛАСОВАНИЕ формирует для вызываемой задачи запрос на вызов ЗАГРУЗЧИКа. В поле В вызываемой

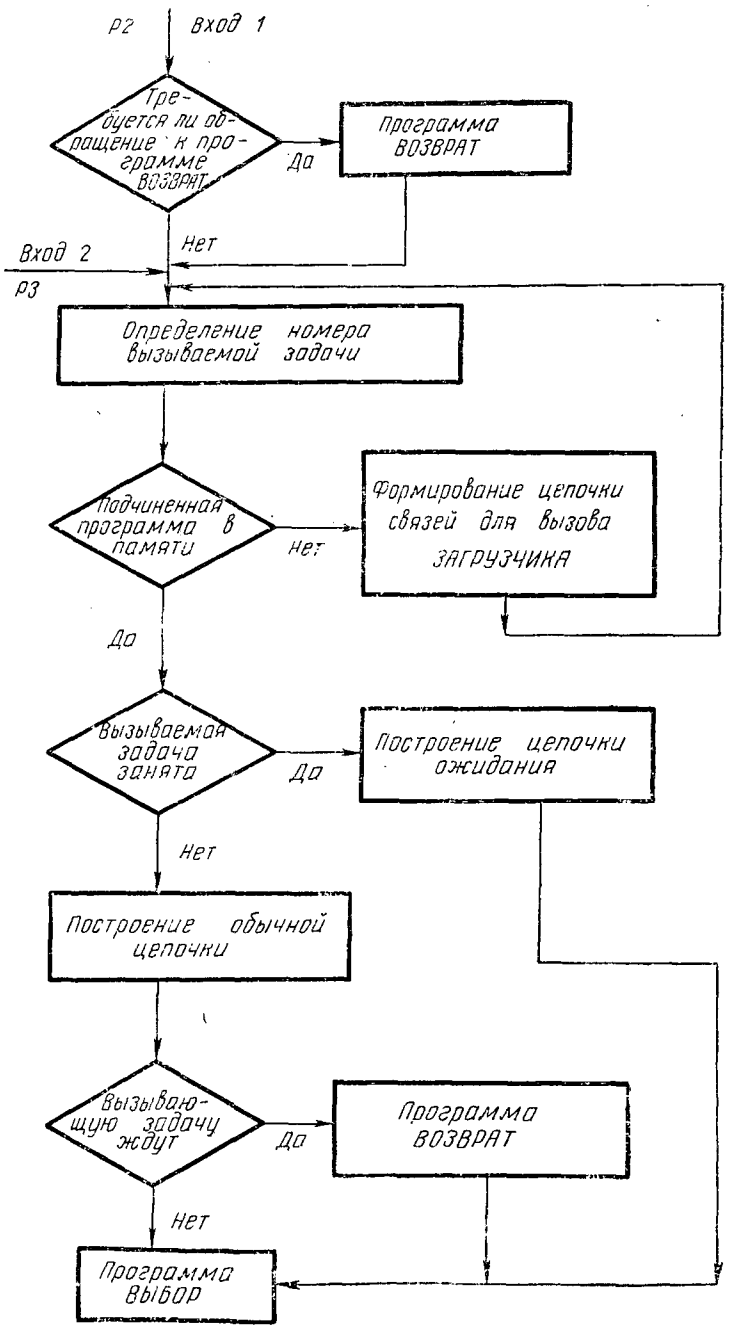


Рис. 12.5. Программа СОГЛАСОВАНИЕ

задачи указывается номер блока управления ЗАГРУЗЧИКа, а в поле Е блока управления ЗАГРУЗЧИКом указывается номер задачи, которая его вызывает. Если к этому моменту времени ЗАГРУЗЧИК уже активен, т. е. его поле Е отлично от нуля, то в вызванной задаче, которая по отношению к ЗАГРУЗЧИКу является вызывающей, вместо поля В заполняется поле Г, а в блоке управления ЗАГРУЗЧИКа в поле Ж добавляется единица. Управление передается на вход 2. При этом параметры не меняются, продолжается повторный анализ той же пары вызываемой и вызывающей программ. В этом случае в поле Д вызываемой программы будет указан адрес, начиная с которого транзитная программа будет вызвана в основную память.

Если программа, подчиненная вызываемой задаче, находится в основной памяти, то поле Д не равно нулю и указывает начальный адрес подчиненной программы. Вызываемая программа может находиться в основной памяти, если она резидентная или если она транзитная, но уже была вызвана какой-либо другой задачей. Поэтому анализируется значение поля Е ее блока управления. Если значение поля Е вызываемой задачи отлично от нуля, то вызываемая задача занята, и для вызывающей ее задачи необходимо построить цепочку ожидания. В этом случае в поле Г вызывающей задачи записывается номер точки входа в ТАБЛИЦУ ЗАДАЧ вызываемой задачи, а в поле Ж вызываемой задачи добавляется единица для увеличения счетчика ожидающих задач. Управление передается программе ВЫБОР.

Если поле Е вызываемой задачи равно нулю, то устанавливается обычная цепочка: номер точки входа вызываемой задачи размещается в поле В вызывающей задачи, а номер точки входа вызывающей задачи размещается в поле Е вызываемой задачи.

Согласование цепочек связей между вызываемой и вызывающей задачами окончено и можно было бы передать управление программе ВЫБОР, которая определила бы задачу с высшим приоритетом, которой нужно передать управление. Однако бывают ситуации, когда вызывающую задачу вызывает какая-либо другая задача и при этом требуется, чтобы вызываемой задаче подчинялась другая программа *Р2*.

Рассмотрим пример. Пусть параметрами к программе СОГЛАСОВАНИЕ будут номера задач М1 и М2. Для задачи М1 должна быть выполнена программа, подчиненная задаче М2. К этому же времени программы *Р3*, управляющие вызовом транзитов *Р2* в основную память, определили, что задачу М1 вызывает задача М3. Но задача М1 может быть активной только в том случае, если ее вызвала какая-либо задача, например М4. При этом задача М1 должна выполнить для задачи М4 программу *Р2* с номером 42. Задача М3 вызвала задачу М1 для выполнения программы 74. Программы *Р3* управления транзитами сохранят такой вызов в ТАБЛИЦЕ ОЖИДАНИЯ. Каждый вход в ТАБЛИЦУ ОЖИДАНИЯ состоит из двух байтов. Первый байт содержит номер программы, подчинен-

ной вызываемой задаче, в нашем примере 74, второй байт — номер вызывающей задачи, в нашем примере М3.

Таблица 12.4

№ п/п	А	Б	В	Г	Д	Е	Ж	З	И
0									
1									
2									
⋮									
М1			М2			М4			
⋮									
М2						М1			
⋮									
М3									
⋮									
М4			М1						
⋮									
	дополнительная точка входа								

Программа СОГЛАСОВАНИЕ в нашем примере построит цепочку связей для задач М1, М2, как это показано в табл. 12.4. Программы *Р3* управления транзитами заполнят вход в ТАБЛИЦЕ ОЖИДАНИЯ.

При этом возможно, что задача М1 закончила свои действия для задачи М4 и может выполняться для задачи М3. Но для этого необходимо установить цепочку связей в ТАБЛИЦЕ ЗАДАЧ для задач М1 и М3.

Поэтому после установления обычной цепочки связей программа СОГЛАСОВАНИЕ проверяет по ТАБЛИЦЕ ОЖИДАНИЯ, нет ли задачи, ожидающей вызывающую задачу. Если таких задач нет, то между всеми задачами согласованы связи, и управление передается программе ВЫБОР.

Если ожидающая задача есть, то для вызывающей задачи осуществляется обращение к программе ВОЗВРАТ для проверки, не окончила ли вызывающая задача свои действия. Если она окончи-

ла выполнение, то программа ВОЗВРАТ очистит ее цепочку связей в ТАБЛИЦЕ ЗАДАЧ и установит новую цепочку для связи с ожидающей ее задачей, в нашем примере М3. После этого управление будет передано программе ВЫБОР.

12.5. Программа выбор

Все выполненные системные задачи передают управление программе ВОЗВРАТ посредством использования макрокоманды обращения к СУПЕРВИЗОРу SVC 32. Аналогично все программы *Р*3, закончившие свои действия, передают управление программе ВЫБОР, которая также является программой *Р*3. Основное назначение этой программы выбрать или проблемную задачу, или системную задачу с высшим приоритетом, или программу *Р*3, которой надо передать управление. Выбор осуществляется двумя путями: программам *Р*3 управление передается на основании анализа регистра флагов прерываний, задачам — на основании анализа ТАБЛИЦЫ ЗАДАЧ. Программа ВЫБОР является третьей программой, имеющей доступ к ТАБЛИЦЕ ЗАДАЧ. Но в отличие от программ ВОЗВРАТ и СОГЛАСОВАНИЕ ВЫБОР не имеет права модифицировать эту таблицу.

Во время выполнения программ *Р*3 все возникающие причины прерываний запоминаются в регистре прерываний, ожидая обработки, так как процессор в состоянии *Р*3 является непрерываемым. Регистр прерываний сохраняет возникшие причины прерываний в порядке их приоритета и тем самым представляет собой очередь прерываний. Вот почему, когда программы *Р*3 заканчивают свои действия и передают управление программе ВЫБОР, она сначала проверяет, есть ли очередь в регистре флагов прерываний. Блок-схема программы ВЫБОР приведена на рис. 12.6.

Если очередь есть, то процессор переводится в состояние *Р*2, которое является прерываемым. Немедленно происходит прерывание с наивысшим приоритетом. При возникновении прерывания управление получает программа *Р*3 АНАЛИЗ ПРЕРЫВАНИЙ, которая определяет путь обработки данного прерывания.

Если прерываний, ожидающих обслуживания, нет, то программа ВЫБОР начинает анализ ТАБЛИЦЫ ЗАДАЧ для определения задачи, которой следует передать управление. Критерием выбора проблемной задачи является выполнение следующих услозий: значения полей Б, В, Г блока управления данной задачи должны равняться нулю (табл. 12.1 и 12.2). Если значение поля Б проблемной задачи не равно нулю, это означает, что она ждет какого-либо события, например окончания операции ввода-вывода или ответа оператора. Если не равны нулю значения полей В или Г, то проблемная задача вызвала другую задачу и ждет окончания ее действий.

Критерием выбора системной задачи является выполнение следующих условий: значения полей Б, В, Г должны равняться нулю, а значение поля Е должно быть отлично от нуля. Если значение по-

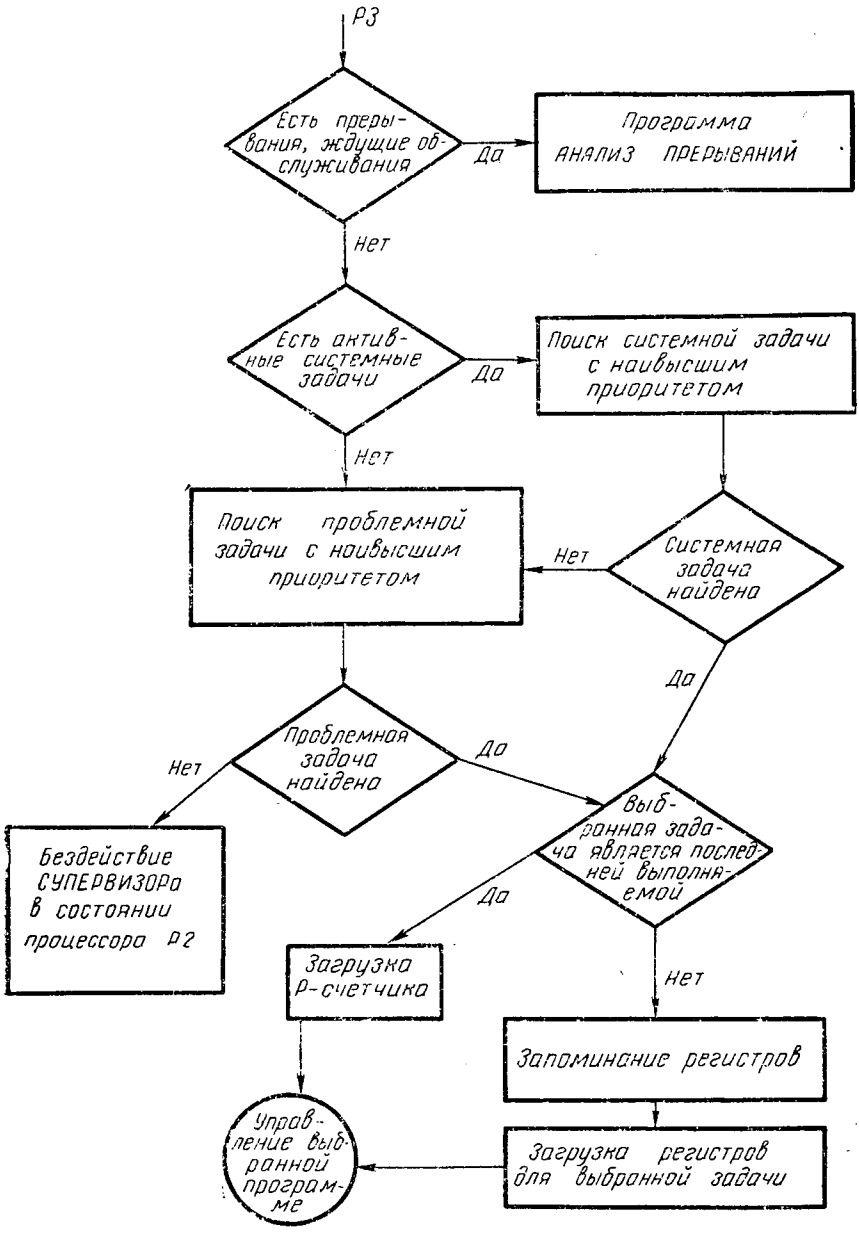


Рис. 12.6. Программа ВЫБОР

ля Е системной задачи равно нулю, то нет задачи, вызывающей данную, и, следовательно, не имеет смысла передавать такой задаче управление. За правильное и своевременное заполнение всех полей ТАБЛИЦЫ ЗАДАЧ отвечают программы СОГЛАСОВАНИЕ и ВОЗВРАТ. Неиспользованные участки в ТАБЛИЦЕ ЗАДАЧ, отведенные проблемным задачам, заполняются единицами, отведенные системным задачам — нулями.

Просмотр ТАБЛИЦЫ ЗАДАЧ осуществляется сверху вниз. Чтобы его сократить, имеется счетчик активных *Р2* задач. Этот счетчик расположен в ТАБЛИЦЕ АДРЕСОВ. Его значение увеличивается на единицу программой СОГЛАСОВАНИЕ, когда она устанавливает обычную цепочку связей между вызывающей и вызываемой задачами, и уменьшается на единицу программой ВОЗВРАТ, когда она очищает цепочку связей для системной задачи, окончившей выполнение. Подобного счетчика для проблемных задач не существует.

Прежде чем начать просмотр ТАБЛИЦЫ ЗАДАЧ, программа ВЫБОР проверяет, равен ли счетчик активных системных задач нулю. Если равен, то программа ВЫБОР сразу переходит к поиску проблемной задачи. Если счетчик не равен нулю, то это не означает, что обязательно существует системная задача, удовлетворяющая критерию выбора. Поэтому начинается последовательный просмотр ТАБЛИЦЫ ЗАДАЧ для поиска системной задачи, значение поля Е которой отлично от нуля. Если найдена такая системная задача, то проверяется, равны ли у нее значения полей Б, В, Г нулю. Если равны, то найденная системная задача считается выбранной. Если не равны, то продолжается поиск следующей системной задачи, у которой значение поля Е отлично от нуля. Если таких системных задач нет, то управление передается на поиск проблемной задачи.

Поиск проблемной задачи осуществляется другим образом. Так как приоритет проблемных задач может изменяться динамически и позиция проблемной задачи в ТАБЛИЦЕ ЗАДАЧ не определяет ее приоритета, то просмотр ТАБЛИЦЫ ЗАДАЧ нельзя осуществлять последовательно. Приоритет проблемных задач определяется по ТАБЛИЦЕ ПРИОРИТЕТОВ ЗАДАЧ, адрес которой находится в ТАБЛИЦЕ АДРЕСОВ. В этой таблице проблемные задачи располагаются в порядке их приоритетов.

Просмотр ТАБЛИЦЫ ЗАДАЧ для поиска проблемной задачи ведется в соответствии с ТАБЛИЦЕЙ ПРИОРИТЕТОВ ЗАДАЧ. Берется первая задача из ТАБЛИЦЫ ПРИОРИТЕТОВ и для нее по ТАБЛИЦЕ ЗАДАЧ проверяется критерий выбора, т. е. проверяется равенство нулю полей Б, В, Г. Если критерий выполнен, то проблемная задача считается выбранной. Если критерий не выполнен, то обрабатывается следующая задача из ТАБЛИЦЫ ПРИОРИТЕТОВ. Если не нашлось проблемной задачи, которой можно передать управление, СУПЕРВИЗОР бездействует в состоянии процессора *Р2*.

Если системная или проблемная задача выбрана, то выполняют-

ся следующие действия. Проверяется, не является ли выбранная задача той, которая выполнялась последней. Номер последней выполняемой задачи хранится в ТАБЛИЦЕ АДРЕСОВ. Если номера выбранной и последней выполняемой задачи совпали, то нет необходимости перезагружать все регистры подчиненной им программы, так как их содержимое никем не испорчено. Достаточно из поля А блока управления выбранной задачи загрузить Р-счетчик. Это равносильно передаче управления выбранной задаче. Если выбранная задача не является последней выполняемой, то производится запоминание всех регистров последней выполняемой задачи, загрузка регистров для выбранной задачи, загрузка Р-счетчика из поля А ТАБЛИЦЫ ЗАДАЧ, и управление передается выбранной задаче.

Содержимое шестнадцати общих регистров и четырех регистров с плавающей запятой проблемной задачи хранится в ОБЛАСТИ ПАМЯТИ СУПЕРВИЗОРа. Эта область предназначена для совместного использования СУПЕРВИЗОРом и проблемной задачей. Она занимает первые 160 байтов области основной памяти, которая отведена проблемной задаче. Структура и назначение этой области будут рассмотрены в главе «УПРАВЛЕНИЕ ЗАДАНИЯМИ».

Содержимое общих регистров, используемых системной задачей, хранится в первых 36 байтах области основной памяти, отведенной этой задаче.

ВОПРОСЫ К ГЛАВЕ

1. Назначение управляющих таблиц.
2. Какая задача будет выбрана для выполнения в ситуации, изображенной в табл. 12.4?
3. Назовите критерий выбора: а) проблемной задачи; б) системной задачи.
4. Какие поля будут заполняться в ТАБЛИЦЕ ЗАДАЧ при установлении: а) обычной цепочки связей; б) цепочки ожидания?
5. Как осуществляется поиск проблемной задачи?
6. Считается ли задача ожидающей, если для нее заполнено: а) поле В; б) поле Г?
7. Какие программы имеют право модифицировать ТАБЛИЦУ ЗАДАЧ?
8. Какой программе передают управление все системные задачи, закончившие выполнение?
9. Какая программа называется подчиненной?
10. Может ли одной задаче подчиняться: а) несколько системных задач; б) несколько проблемных задач?

Глава 13

СУПЕРВИЗОР ВВОДА-ВЫВОДА

13.1. Основные понятия

Рассмотрим группу программ *Р3*, предназначенных для управления операциями ввода-вывода. Некоторые действия, необходимые при выполнении операции ввода-вывода, могут быть выполнены

только в состоянии процессора *P2* или *P3*. К ним относятся обслуживание запуска и окончание операции ввода-вывода, распознавание ошибок устройств. Эти действия осуществляются при возникновении прерываний от ввода-вывода. Поэтому СУПЕРВИЗОР является посредником между машиной и пользователем для обслуживания запросов, связанных с обработкой операций ввода-вывода. Такое посредничество, с одной стороны, обеспечивает эффективное использование внешних устройств, с другой стороны, освобождает программиста от выполнения некоторых рутинных действий по обслуживанию операций ввода-вывода и предоставляет ему наиболее полную информацию о ходе и результатах выполнения операций ввода-вывода.

СУПЕРВИЗОР ВВОДА-ВЫВОДА представляет собой совокупность программ, часто называемых физической системой управления вводом-выводом (PIOCS). Основными функциями физической системы управления вводом-выводом являются:

формирование блока управления данными;

выдача запроса СУПЕРВИЗОРу на выполнение операции ввода-вывода;

обработка информации о результатах выполнения операции.

Программа канала представляет собой цепочку команд канала для каждого типа устройств. Блок управления данными содержит информацию, необходимую СУПЕРВИЗОРу для выполнения операции ввода-вывода. Запрос на выполнение операции ввода-вывода осуществляется с помощью команды обращения к СУПЕРВИЗОРу SVC или с помощью макрокоманд ввода-вывода, макрорасширения которых содержат команду обращения к СУПЕРВИЗОРу. Информация о результатах выполнения операции ввода-вывода помещается СУПЕРВИЗОРом в БЛОК УПРАВЛЕНИЯ ДАННЫМИ по окончании операции и содержит сведения о том, как закончился ввод-вывод. Если операция закончилась аварийно, то информация содержит сведения о характере ошибок и причинах аварийного окончания.

Программист может сам закодировать в своей программе выполнение перечисленных функций с помощью специального набора макрокоманд физической системы управления вводом-выводом. Такой метод работы называется физическим уровнем. Физический уровень программирования ввода-вывода обеспечивает гибкость при работе с внешними устройствами. Его недостаток в том, что программист должен знать специфику работы внешних устройств. Поэтому операционная система предлагает программисту более высокий уровень программирования ввода-вывода — логический уровень. Пользуясь логическим уровнем, программист должен заботиться только о логической структуре своих данных. Все функции, связанные с передачей данных с внешних носителей в основную память или, наоборот, из основной памяти на внешние носители, выполняет операционная система.

При программировании на логическом уровне используются

макрокоманды, объединенные общим названием «логическая система управления вводом-выводом». Логическая система управления вводом-выводом в своей работе опирается на физическую систему управления вводом-выводом.

Физическая система управления вводом-выводом обеспечивает выполнение запросов на операции ввода-вывода для проблемных программ и программ СУПЕРВИЗОРа. Она обеспечивает выполнение макрокоманд логического уровня для методов доступа, предоставляемых программисту, и методов доступа к системному файлу.

Программы физической системы управления вводом-выводом в зависимости от типа выполняемых действий можно разделить на три группы.

1. Программы, осуществляющие начальную подготовку для выполнения операций ввода-вывода. Они объединены под общим названием ПЛАНИРОВЩИК КАНАЛОВ.

2. Программы, управляющие порядком выполнения запросов на операции ввода-вывода. Эти программы объединены под общим названием «диспетчерские программы», или ДИСПЕТЧЕР.

3. Программы, обслуживающие задержки от прерываний ввода-вывода и проверяющие условия выполнения операций ввода-вывода для обнаружения ошибок устройств. Эти программы объединены под общим названием ОКОНЧАНИЕ ВВОДА-ВЫВОДА.

13.2. Управляющие таблицы

Аналогично тому, как управление задачами связано с обработкой некоторых управляющих таблиц, управление вводом-выводом осуществляется также посредством анализа ряда таблиц. Но если управляющие таблицы, на основании которых осуществляется управление задачами, располагаются в области памяти СУПЕРВИЗОРа и обрабатываются только программами СУПЕРВИЗОРа, то таблицы, используемые для управления вводом-выводом, могут располагаться как в области основной памяти проблемных задач, так и в области СУПЕРВИЗОРа. При этом к таблицам, расположенным в области проблемных задач, можно обратиться из подчиненной программы. Такое обращение с таблицами СУПЕРВИЗОРа ВВОДА-ВЫВОДА дает программисту гибкий аппарат управления файлами данных.

Информация, необходимая для выполнения операции ввода-вывода для проблемной задачи, содержится в ТАБЛИЦЕ ОПРЕДЕЛЕНИЯ ФАЙЛА, которая размещается в теле проблемной программы в области, предназначенной для связи с СУПЕРВИЗОРом ВВОДА-ВЫВОДА. Адрес этой таблицы передается как параметр в команде обращения к СУПЕРВИЗОРу.

Формирование ТАБЛИЦЫ ОПРЕДЕЛЕНИЯ ФАЙЛА (ТОФ) происходит при выполнении одной из макрокоманд определения файлов. Макрокоманды определения файлов DTF... являются описательными командами и задают тип файла, метод организации

и некоторую другую информацию. В операционной системе ИБМ/360 такой командой является DCB, в дисковой операционной системе Единой системы ЭВМ и в операционной системе Джей Системы-4 имеется несколько макрокоманд, начинающихся с аббревиатуры DTF... Построение таблицы завершается при выполнении макрокоманды OPEN (открыть).

ТАБЛИЦА ОПРЕДЕЛЕНИЯ ФАЙЛА делится на две части. Первые сорок байтов этой таблицы называются БЛОКОМ УПРАВЛЕНИЯ ДАННЫМИ (БУД). Структура блока управления данными одинакова для файлов с различной организацией и не зависит от устройства, с которым связана обработка файла. Оставшаяся часть таблицы определения файла существенно зависит от используемого метода доступа, от типа устройства и поэтому нами рассматриваться не будет. Отметим только, что во второй части таблицы содержатся значения различных счетчиков, различные флаги и маркеры, а также специфическая информация для каждого типа устройства. Структура блока управления данными приведена в табл. 13.1.

Таблица 13.1

Байты	Содержание
0—6	Символическое имя таблицы определения файлов
7	Флаги ошибок
8—11	Адрес слова команды канала
12—13	Адрес блока управления главой для системного файла
14—15	Номер входа в таблицу информации об устройствах
16—27	Содержимое регистров, используемых в командах
36—39	канала
28—31	Резервируется
32—36	Флаги окончания операций ввода-вывода

Рассмотрим назначение отдельных полей блока управления данными. Символическое имя ТАБЛИЦЫ ОПРЕДЕЛЕНИЯ ФАЙЛА играет роль идентификатора файла, для которого сформирована данная таблица. Для системного файла в этом поле указывается идентификатор основной главы.

В байте 7 устанавливаются флаги ошибок, обнаруженные во время инициирования операции ввода-вывода.

Адрес слова команды канала (для диска этот адрес будет указывать первую команду канала из цепочки команд) может быть указан программистом, если он желает программировать ввод-вывод на физическом уровне. При программировании ввода-вывода на логическом уровне этот адрес будет сформирован СУПЕРВИЗОРом на основании макрокоманды определения файла. Выполне-

ние любой другой макрокоманды ввода-вывода в проблемной программе вызовет обращение к СУПЕРВИЗОРу. СУПЕРВИЗОР перешлет адрес слова команды канала в очередь канала для сохранения. Во время инициации операции ввода-вывода этот адрес будет переслан в постоянно фиксируемую область основной памяти. Когда канал начнет перемещение данных, адрес своих команд он выберет из этой области.

Байты 12—13 проблемными программами не используются. Они предназначены для указания адреса блока управления главой системного файла.

Байты 14—15 служат для опознавания медленных устройств или магнитной ленты по таблице информации об устройствах. Эта информация устанавливается макрокомандой ОТКРЫТЬ (OPEN).

Байты 16—39 используются СУПЕРВИЗОРом, чтобы информировать проблемную программу, как закончилась операция ввода-вывода. Они включают содержимое различных регистров канала, слово состояния канала и флаги аварийного окончания операции ввода-вывода.

Блок управления данными доступен проблемной программе, которая может провести анализ окончания операции ввода-вывода, если это необходимо.

Число внешних устройств, их назначение и характеристики могут меняться от одной конфигурации вычислительной машины к другой. Однако СУПЕРВИЗОР должен иметь информацию о каждом устройстве, которое может потребоваться проблемной программе. Информация об устройствах содержится в ТАБЛИЦЕ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ. ТАБЛИЦА ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ (ТИУ) формируется в момент генерации операционной системы и размещается в области программ *РЗ*. Для каждого устройства формируется один вход в таблицу. Таблица будет содержать столько входов, сколько внешних устройств имеется в данной конфигурации вычислительной машины. Каждый вход содержит всю необходимую информацию о физических характеристиках данного устройства. Описание для одного входа занимает 26 байтов и приведено в табл. 13.2.

Таблица 13.2

Байты	Содержание
0—2	Мнемоническое имя устройства
3—4	Тип устройства
5	Флаги
6—7	Адрес устройства в главном канале
8—9	Адрес заголовка очереди к главному каналу
10—11	Адрес устройства в альтернативном канале
12—13	Адрес заголовка очереди к альтернативному каналу

Продолжение табл. 13.2

Байты	Содержание
14—19	Адреса таблицы назначения физических устройств логическим устройствам
20	Номер задачи
21—22	Маркеры
23—25	Адрес таблицы определения файла

Рассмотрим назначение отдельных полей таблицы. Мнемоническое имя устройства представляет собой имя логического устройства, которое будет использоваться в сообщениях оператору, например, в случае работы этого устройства.

В байтах 3—4 указывается уникальный идентификатор данного устройства и номер его модели. Он используется программами распознавания ошибок для определения действий, которые необходимо предпринять при неправильной работе этого устройства.

В байте 5 указаны флаги некоторых специфических назначений устройства (например, устройство зарезервировано или находится в режиме автономной работы, или отключено).

Адрес устройства состоит из двух компонентов: номера канала, к которому присоединено устройство, и собственного номера устройства. В байтах 6—7 указывается адрес устройства, которое подсоединено к главному каналу. Так как каждое устройство может быть подсоединено к двум и более каналам, то в байтах 10—11 указывается адрес этого же устройства, но подсоединенного ко второму каналу. Второй канал обычно называют альтернативным каналом.

В байтах 8—9 содержится адрес заголовка очереди к главному каналу, а в байтах 12—13 — адрес заголовка очереди к альтернативному каналу. Если альтернативный канал не используется, т. е. устройство подсоединено всегда к одному каналу, то в байтах 10—11 и 12—13 содержатся нули.

Байты 14—19 устанавливаются во время работы системы. Это поле используется для томов, расположенных на магнитных лентах и устройствах с прямым доступом. В этом поле указывается адрес таблицы соответствия физических устройств логическим устройствам. При этом для магнитных лент и устройств с прямым доступом формируются различные таблицы. Для тома с системным файлом в этом поле устанавливаются специальные маркеры.

Байт 20 устанавливается программой ГЛАВНЫЙ ПЛАНИРОВЩИК во время функционирования системы. Для медленных устройств и магнитных лент в этом байте указывается номер задачи (т. е. номер точки входа в ТАБЛИЦУ ЗАДАЧ), которой выделено это устройство. Для устройства с прямым доступом в этом байте указывается число файлов, помещаемых на данном устройстве. Это число будет определено на основании макрокоманд ОТКРЫТЬ

(OPEN), которые потребуют выделения томов для указанных в них файлов.

Байты 21—22 устанавливаются программой ГЛАВНЫЙ ПЛАНИРОВЩИК для указания особых ситуаций. Например, для устройств с прямым доступом устанавливается маркер, если это устройство еще не открыто. Для медленного устройства указывается номер потока, которому оно выделено.

Байты 23—25 устанавливаются при выполнении макрокоманды ОТКРЫТЬ (OPEN). В них указывается адрес таблицы определения того файла, который будет обрабатываться на этом устройстве.

Мы уже знаем, что существуют два типа каналов: селекторные и мультиплексные. К селекторному каналу подключается одно устройство. Хотя селекторный канал выполняет перемещение данных за единицу времени от одного устройства, он может обслуживать несколько файлов из различных задач. Это значит, что запросы на операцию ввода-вывода, выполняемые одним селекторным каналом, должны быть установлены в очередь и задачи, потребовавшие операцию ввода-вывода, обслуживаются не немедленно, а должны ждать, пока освободится селекторный канал. Сведения об ожидающих задачах собираются в очереди канала. Для каждого селекторного канала существует одна очередь. Максимальное число входов в эту очередь определяется во время генерации системы. Оно не может быть меньше числа файлов, которые могут быть одновременно открыты макрокомандой ОТКРЫТЬ (OPEN) на устройстве, которое обслуживает селекторный канал. Мультиплексный канал может одновременно перемещать данные от нескольких устройств. Поэтому очередь канала создается для каждого устройства, которое обслуживается этим мультиплексным каналом. Один мультиплексный канал имеет столько очередей, сколько устройств он обслуживает.

Очередь канала состоит из заголовка и входов в очередь. Заголовок занимает одно слово. Структура заголовка представлена в табл. 13.3.

Таблица 13.3

Байты	Содержание
0	Число всех входов в очередь
1	Число текущих входов в очередь
2	Флаги
3	Счетчик автономной работы

В байте «нуль» содержится число всех входов в очередь канала. Оно является тем числом, которое определяется во время генерации системы для селекторного канала и равно единице для мультиплексного канала. В байте «один» содержится число текущих вхо-

дов, т. е. фактическое число входов в очередь канала в текущий момент времени. Это число увеличивается на единицу, когда вход располагается в очереди канала, и уменьшается на единицу, когда перемещение окончено и вход вычеркивается из очереди. Если задача запросит операцию ввода-вывода в тот момент, когда значение байта нуль равно значению байта один, то очередь считается переполненной, задача определяется как ожидающая (в ТАБЛИЦЕ ЗАДАЧ в ее блоке управления записывается условие ожидания и запоминается значение Р-счетчика). Задача будет ждать до тех пор, пока не освободится вход в очереди канала. Этим и определяется требование, чтобы максимальное число входов в очередь селекторного канала было не меньше числа файлов, которые могут быть одновременно открыты для селекторного канала.

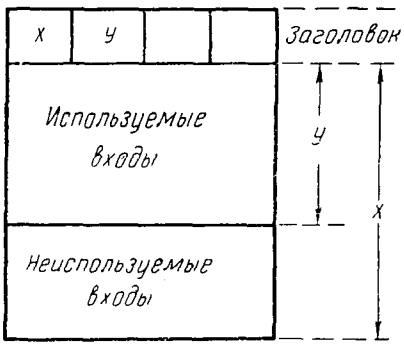


Рис. 13.1. Структура очереди к селекторному каналу

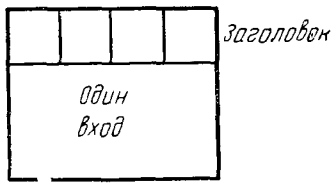


Рис. 13.2. Структура очереди к мультиплексному каналу

В байте «два» устанавливаются флаги, которые определяют состояние очереди, например, как прошла выполняемая операция ввода-вывода и можно ли вычеркнуть вход из очереди, переполнена ли очередь.

В байте «три» находится счетчик, в котором накапливаются сведения о том, сколько раз канал работал в автономном режиме. Например, для магнитной ленты в этом байте будет указано число перемоток.

Таким образом, каждый селекторный канал имеет одну очередь, структура которой приведена на рис. 13.1. Число очередей к мультиплексному каналу будет равно числу устройств, подсоединенных к этому каналу. Каждая очередь мультиплексного канала будет состоять из одного входа. Структура очереди к мультиплексному каналу приведена на рис. 13.2.

Все очереди канала относятся к программам *РЗ* и располагаются в области управляющей программы.

Структура каждого входа в очередь канала представлена в табл. 13.4.

Таблица 13.4

Байты	Содержание
0—3	Адрес таблицы определения файла
4—5	Адрес устройства в главном канале
6—7	Адрес заголовка альтернативной очереди канала
8	Ключ защиты
9	Номер задачи
10—13	Адрес слова команды канала

В байтах 0—3 содержится адрес ТАБЛИЦЫ ОПРЕДЕЛЕНИЯ ФАЙЛА, точнее, адрес БЛОКА УПРАВЛЕНИЯ ДАННЫМИ. Этот адрес является параметром в команде обращения к СУПЕРВИЗОРу, которую вырабатывает макрокоманда ввода-вывода.

В байтах 4—5 содержится адрес устройства (мы помним, что он состоит из номера канала и номера самого устройства), которое должно выполнить требуемую операцию ввода-вывода. СУПЕРВИЗОР помещает эту информацию из ТАБЛИЦЫ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ, адрес которой он узнает из БЛОКА УПРАВЛЕНИЯ ДАННЫМИ.

В байтах 6—7 содержится адрес заголовка альтернативной очереди, который берется также из ТАБЛИЦЫ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ. Если альтернативной очереди нет, то в этих байтах содержатся нули.

В байте 8 содержится ключ защиты основной памяти, куда (или откуда) будет выполняться перемещение данных. Ключ защиты памяти берется из ТАБЛИЦЫ ЗАДАЧ блока управления той задачи, которая запросила операцию ввода-вывода. В байте 9 указывается номер задачи, для которой выполняется операция ввода-вывода.

Байты 10—13 содержат адрес первого слова команды канала из цепочки команд. Этот адрес берется из БЛОКА УПРАВЛЕНИЯ ДАННЫМИ проблемной задачи. При инициации операции ввода-вывода этот адрес будет размещен в постоянно фиксированной области основной памяти.

На рис. 13.3 приведена схема взаимосвязи рассмотренных нами таблиц, когда выполняется операция ввода-вывода медленным устройством.

Макрокоманда ввода-вывода, выполненная в проблемной программе, сформирует команду обращения к СУПЕРВИЗОРу, одним из параметров которой будет адрес ТАБЛИЦЫ ОПРЕДЕЛЕНИЯ ФАЙЛОВ. СУПЕРВИЗОР выберет из блока управления данными (вспомним, что это первые сорок байтов ТАБЛИЦЫ ОПРЕДЕЛЕНИЯ ФАЙЛОВ) номер входа в ТАБЛИЦУ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ. По ТАБЛИЦЕ ИНФОРМАЦИИ ОБ УСТРОЙСТ-

ВАХ СУПЕРВИЗОР определит адрес заголовка очереди канала и сформирует вход в очередь канала на требуемую операцию. Информацию для организации входа СУПЕРВИЗОР почерпнет из ТАБЛИЦЫ ОПРЕДЕЛЕНИЯ ФАЙЛОВ, ТАБЛИЦЫ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ и ТАБЛИЦЫ ЗАДАЧ.

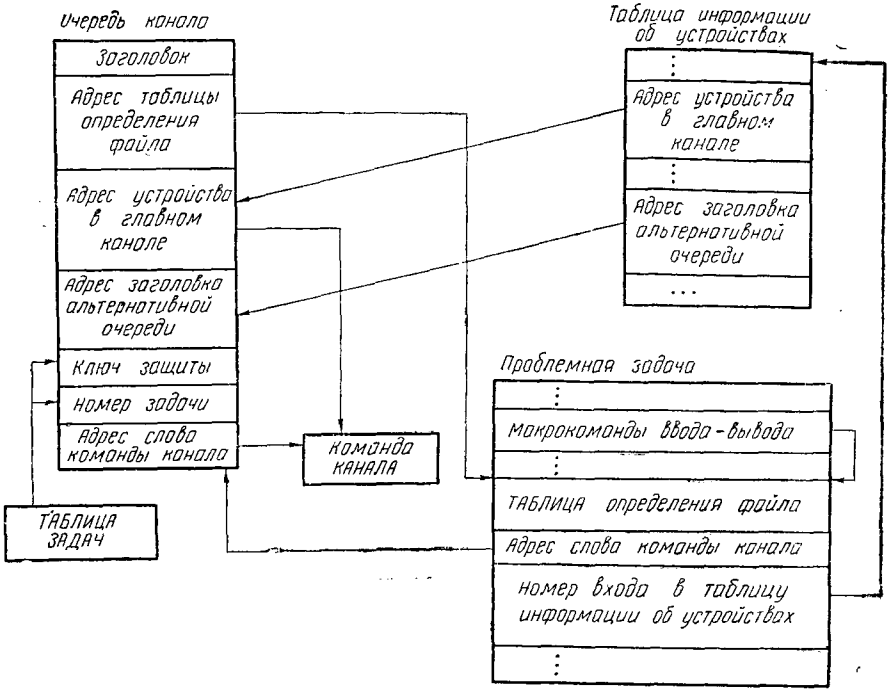


Рис. 13.3. Схема взаимосвязи таблиц

Когда СУПЕРВИЗОР начнет инициировать операцию ввода-вывода, он переместит адрес слова команды канала из очереди канала в байты 72—75 постоянно фиксируемой области памяти и выполнит команду НАЧАТЬ ВВОД-ВЫВОД. По этой команде управление получает канал, который начинает работать параллельно с работой процессора. Происходит прерывание от ввода-вывода, которое СУПЕРВИЗОР обрабатывает обычным образом. Канал выберет из слова адреса канала адрес первой команды цепочки команд канала и начнет выполнять эти команды. По командам канала устройство начнет перемещение данными. По окончании перемещения вырабатывается сигнал окончания операции ввода-вывода, вновь произойдет прерывание от ввода-вывода, и СУПЕРВИЗОР получит управление для обработки этой причины прерывания.

Отличительной чертой таблиц, которые требуются для выполнения операций ввода-вывода медленными устройствами и магнит-

ными лентами, является то, что каждым устройством обслуживается один файл. В отличие от них устройство с прямым доступом может содержать несколько различных файлов, которые могут быть одновременно доступны нескольким задачам, при этом одна из задач может потребовать монопольное управление томом или файлом. Как известно, файл — совокупность записей, том — стандартный блок внешней памяти. На одном томе могут располагаться несколько файлов или один файл может занимать несколько томов.

Если задача получила право монопольного управления томом (или файлом), то она запрещает пользоваться этим томом (файлом) до тех пор, пока не закончит свою работу или пока не потеряет право монопольного управления.

Чтобы указать всю требуемую информацию для выполнения подобных функций, необходимо иметь промежуточную таблицу для управления файлами на устройстве с прямым доступом. Такой таблицей является БЛОК УПРАВЛЕНИЯ ФАЙЛАМИ (БУФ). Для доступа к файлу на устройстве с прямым доступом уже не требуется устанавливать связь между ТАБЛИЦЕЙ ОПРЕДЕЛЕНИЯ ФАЙЛОВ и ТАБЛИЦЕЙ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ. Эту роль также выполняет блок управления файлами.

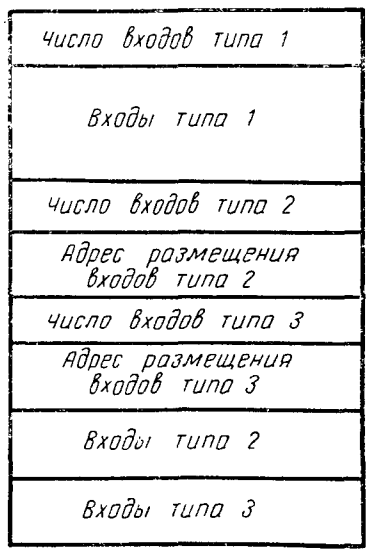


Рис. 13.4. Структура блока управления файлами

БЛОК УПРАВЛЕНИЯ ФАЙЛАМИ строится для каждого файла на устройстве с прямым доступом во время выполнения макрокоманды ОТКРЫТЬ (OPEN) для этого файла. Структура БЛОКА УПРАВЛЕНИЯ ФАЙЛАМИ приведена на рис. 13.4.

Имеются три типа входов в БЛОК УПРАВЛЕНИЯ ФАЙЛАМИ. Каждый файл, для которого выполнена макрокоманда ОТКРЫТЬ (OPEN), обязательно имеет один вход типа «один». Благодаря этому файл включается в состав системного файла и становится доступным. Структура входа типа «один» приведена в табл. 13.5.

Байт нуль содержит маркеры, определяющие состояние файла, например право монопольного управления файлом или томом, маркер системного файла. При этом право монопольного управления этим файлом могут иметь как системные, так и проблемные задачи. Право монопольного управления томом могут иметь только системные задачи. Для проблемных задач этот маркер всегда устанавливается в нуль.

Таблица 13.5

Байты	Содержание
0	Маркеры
1	Счетчик активных перемещений
2—4	Адрес ТАБЛИЦЫ ОПРЕДЕЛЕНИЯ ФАЙЛА
5—6	Адрес входа типа 3
7	Номер задачи, обращающейся к данному файлу
8	Флаги для данной задачи
9—11	Адрес блока управления главами
12—13	Номер входа в блок управления главами
14—15	Резервируются
16—17	Начальный адрес СПИСКА БИТОВ
18—19	Конечный адрес СПИСКА БИТОВ
20	Адрес таблицы соответствия логических номеров томов
21—23	Резервируются
24—25	Начальный адрес указателя памяти
26—27	Конечный адрес указателя памяти
28—29	Число свободных дорожек
30—37	Информация по первому измерению
38—39	Нуль или адрес входа типа 2

В байте 1 хранится счетчик активных перемещений. Содержание счетчика увеличивается на единицу всякий раз, когда операция ввода-вывода для этого файла прошла успешно, значение счетчика используется при анализе функционирования системы и дисков.

В байтах 2—4 указывается адрес ТАБЛИЦЫ ОПРЕДЕЛЕНИЯ ФАЙЛА. Как и для любого файла, для файла на диске в проблемной программе формируется ТАБЛИЦА ОПРЕДЕЛЕНИЯ ФАЙЛА, связь с которой осуществляется посредством содержимого данного поля.

В байте 7 указывается номер задачи, которой стал доступен этот файл. Таблица определения файла, адрес которой указан в байтах 2—4, принадлежит этой задаче. В байтах 5—6 указывается адрес входа типа «три», который порожден файлом для этой задачи.

Информация, которая содержится в байтах 9—29, зависит от того, установлен ли в единицу маркер системного файла в байте нуль. Если описываемый файл не является системным файлом, маркер устанавливается равным нулю. В этом случае в байтах 9—15 записывается информация, аналогичная хранящейся в байтах 2—8, но уже для другой задачи, которая также использует этот файл. В байтах 16—22 записывается аналогичная информация для третьей задачи, использующей этот файл, в байтах 23—29 — для четвертой.

Размеры входа типа 1 позволяют сохранить информацию о файле для четырех задач. Это является ограничением на число задач, которые могут одновременно пользоваться одним и тем же файлом на диске.

Если маркер системного файла установлен в единицу, то в байтах 9—11 указывается адрес блока управления главой. Блок управления главой аналогичен блоку управления данными. Его структура и назначение будут нами рассмотрены при изучении методов доступа к системному файлу.

В байтах 12—13 указывается номер входа в блок управления главами. В байтах 16—17 указывается начальный адрес расположения в основной памяти СПИСКА БИТОВ. В байтах 18—19 — конечный адрес этого списка. В байте 20 стоит адрес таблицы соответствия логических номеров томов, в которой определены назначения физических устройств логическим томам. В байтах 24—27 хранятся начальный и конечный адреса УКАЗАТЕЛЯ ПАМЯТИ. В байтах 28—29 хранится число свободных дорожек на данном томе.

Файл может занимать один или несколько томов. Количество занимаемых томов определяет размерность файла. Например, файл, занимающий один том, является одномерным, а информация о расположении его на этом томе называется информацией по первому измерению. Файл, занимающий два тома, является двумерным и имеет уже информацию как по первому измерению, так и по второму. Информация по первому измерению располагается всегда в байтах 30—37 входа типа «один» и первоначально содержится во входе типа «три». Информация по второму измерению располагается во входе типа «два».

В байтах 30—37 хранится информация по первому измерению этого файла, указывающая следующие данные: в байтах 30—31 хранится адрес тома в таблице информации об устройствах; в байтах 32—33 указывается абсолютный номер цилиндра, занимаемого данным файлом; в байтах 34—35 указывается абсолютный номер первой дорожки, занимаемой файлом на этом цилиндре; в байтах 36—37 хранится число выделенных файлу дорожек.

В байтах 38—39 стоит нуль, если файл является одномерным, или начальный адрес информации о входах типа «два», если файл — многотомный.

Таблица 13.6

Байты	Содержание
0—7	Относительный адрес расположения файла на томе
8—15	Маска файла
16—23	Построенная цепочка команд канала
24—31	Абсолютный адрес расположения файла на диске

Так как к файлу одновременно могут обратиться несколько задач, при этом одна может требовать чтения этого файла, а другая — записи в файле, то для каждой задачи, использующей этот файл, строится собственный вход типа «три». Адрес этого входа указы-

вается в байтах типа 5—6 входа типа «один». Вход типа «три» блока управления файлами содержит информацию по первому измерению данного файла для данной задачи. Структура входа типа «три» приведена в табл. 13.6.

Вход типа «три» содержит последовательность слов команд канала для обращения к диску, маску файла и информацию о выполняемых перемещениях. Обращение к файлу всегда начинается с адресации его первого измерения. СУПЕРВИЗОР преобразует относительный адрес расположения файла на томе, хранящийся в байтах 0—7 входа типа «три», в абсолютный и запоминает его в байтах 24—31. В байтах 8—15 устанавливается маска файла. Установленная маска может, например, запретить запись информации в файл, если он был открыт для чтения. В маске также содержится информация о том, что необходимо перейти к обработке файла по второму измерению, если файл многотомный.

В байтах 16—23 строится цепочка команд канала для обращения к диску на основании информации из БЛОКА УПРАВЛЕНИЯ ДАННЫМИ для данной задачи.

Если файл многотомный, то для каждого следующего измерения строится вход типа «два». Структура входа типа «два» приведена в табл. 13.7. Вход типа «два» содержит информацию о томе, на котором расположено продолжение файла, аналогичную той, которая хранится в байтах 30—37 входа типа «один».

Таблица 13.7

Байты	Содержание
0—1	Адрес тома в таблице информации об устройствах
2—3	Абсолютный номер цилиндра данного тома
4—5	Абсолютный адрес первой дорожки данного типа
6—7	Число дорожек в этом томе
8—9	Нуль или адрес следующего входа типа «два» для следующего измерения

В байтах 0—1 указывается адрес тома в ТАБЛИЦЕ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ. В байтах 2—3 — абсолютный номер цилиндра данного тома; в байтах 4—5 — абсолютный адрес первой дорожки в этом цилиндре. В байтах 6—7 хранится число дорожек этого цилиндра, выделенных под данный том. В байтах 8—9 стоит нуль, если файл не имеет больше измерений, или номер входа типа «два» для следующего измерения.

13.3. Планировщик каналов

Макрокоманды EXCP, EXCPW обслуживают запросы на ввод-вывод для медленных устройств и магнитных лент. Макрокоманды

EXDP и EXDPW обслуживают запросы на ввод-вывод для устройств с прямым доступом. Имена этих макрокоманд являются аббревиатурой от execute channal program. Группу программ, соответствующих этим макрокомандам, будем называть ПЛАНИРОВЩИК КАНАЛОВ.

По назначению и выполняемым функциям программы, входящие в ПЛАНИРОВЩИК КАНАЛОВ, делятся на две группы: первоначальный анализ и планирование работы каналов. Вход в программы первой группы осуществляется посредством макрокоманд физической системы управления вводом-выводом. Вход в программы второй группы происходит только из программ первой группы.

Назначение программ первой группы, обслуживающих запросы на операции ввода-вывода от медленных устройств или от устройств с прямым доступом, одинаковое, однако для реализации своих функций они используют различные таблицы. Программы первоначального анализа, обслуживающие медленные устройства и магнитные ленты, не используют блока управления файлами.

Основными функциями программ первоначального анализа является проверка правильности параметров; открыт ли файл, к которому осуществляется доступ; выделено ли требуемое устройство задаче для выполнения операции ввода-вывода.

Рассмотрим принципиальную схему действия программ первоначального анализа, предназначенных для обслуживания медленных устройств. Функциональная блок-схема программы ПЕРВОНАЧАЛЬНЫЙ АНАЛИЗ приведена на рис. 13.5.

Единственным параметром макрокоманд, позволяющих войти в ПЛАНИРОВЩИК КАНАЛОВ, является адрес ТАБЛИЦЫ ОПРЕДЕЛЕНИЯ ФАЙЛОВ. На основании этого адреса осуществляется доступ в ТАБЛИЦУ ОПРЕДЕЛЕНИЯ ФАЙЛА, формируется адрес ТАБЛИЦЫ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ. После этого производится проверка, правильно ли сформирован адрес ТАБЛИЦЫ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ. Если неправильно, то в БЛОКЕ УПРАВЛЕНИЯ ДАННЫМИ устанавливаются флаги соответствующей ошибки, и управление передается программе ВЫБОР. Если адрес правильный, то проверяется, не использовалась ли макрокоманда обслуживания медленных устройств для обслуживания файлов на устройствах с прямым доступом. Если это так, то выполняется макрокоманда обращения к СУПЕРВИЗОРу для вызова программ *P2* распознавания ошибок. Если макрокоманда использовалась по назначению, проверяется, системная или проблемная задача запросила операцию ввода-вывода. Если запрос произошел от проблемной задачи, то необходимо проверить, открыт ли файл, к которому осуществляется доступ. Если файл не открыт, то в блоке управления данными устанавливаются соответствующие флаги, и управление передается программе ВЫБОР. Если файл открыт, то необходимо проверить, закончено ли обслуживание запроса или нет. Если закончено, то управление передается программе ВЫБОР, если не закончено, производится анализ очереди канала.

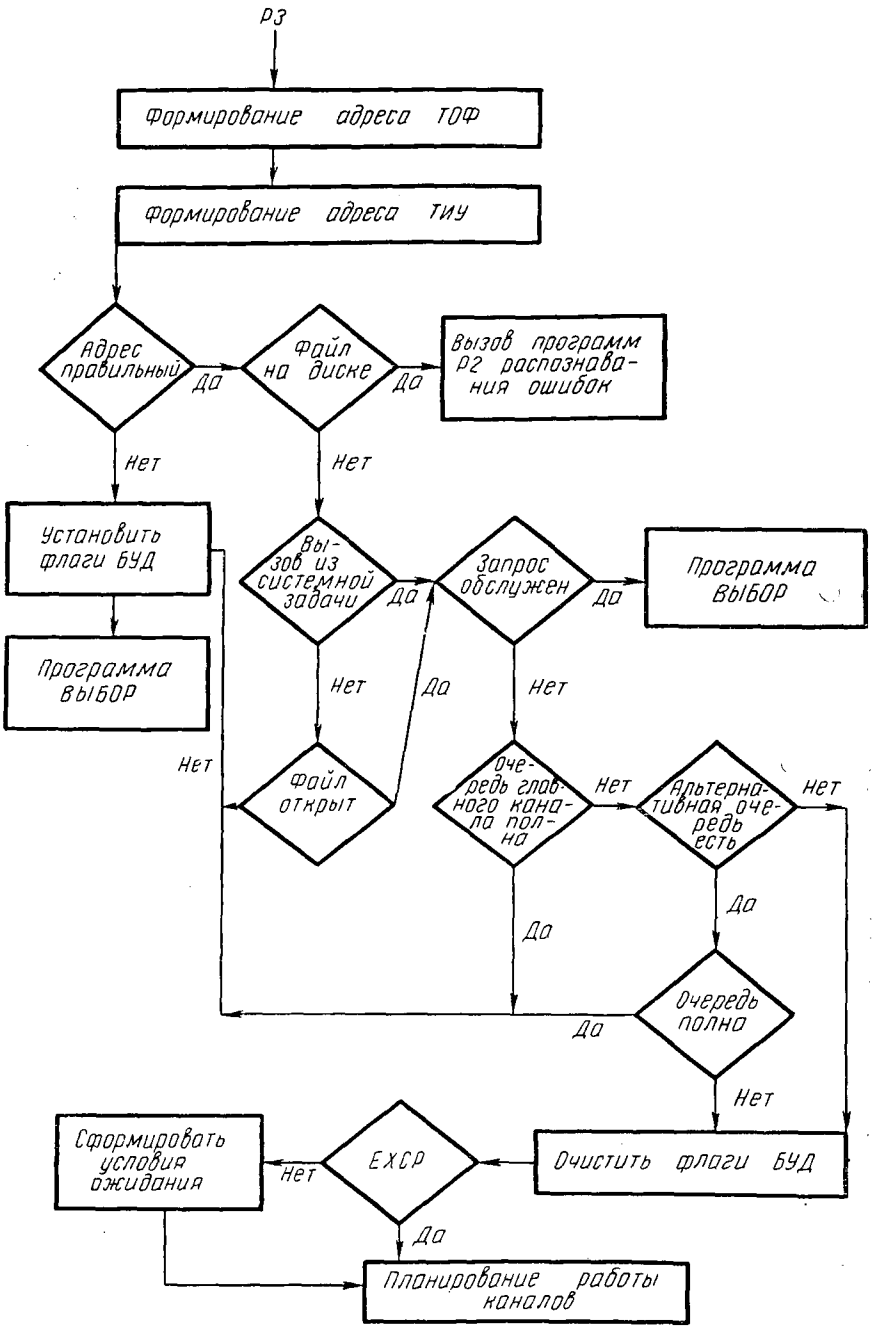


Рис. 13.5. Первоначальный анализ

Так как одно и то же устройство может быть подсоединено к двум каналам, то для него устанавливаются две очереди канала. Одна из этих очередей считается главной, вторая — альтернативной. Анализ начинается всегда с главной очереди на основании флагов, установленных в заголовке очереди.

Если главная очередь канала полна, то в заголовке устанавливается флаг переполнения очереди, в блоке управления задачей в ТАБЛИЦЕ ЗАДАЧ в поле Б устанавливается условие ожидания, добавляется единица в счетчик ожидающих программ, запоминается Р-счетчик программы, для которой осуществился запрос на выполнение операции ввода-вывода, и управление передается программе ВЫБОР.

Если главная очередь к каналу не полна, то проверяется, имеется ли альтернативная очередь к каналу. Если имеется, то проверяется, не полна ли эта очередь. Если очередь полна, то выполняются те же действия, что и при переполнении главной очереди. Если альтернативная очередь не полна, то очищаются соответствующие флаги в блоке управления данными. Затем проверяется, с помощью какой макрокоманды осуществляется доступ к ПЛАНИРОВЩИКУ КАНАЛОВ. Если с помощью макрокоманды EXCP, то управление передается программе ПЛАНИРОВАНИЕ РАБОТЫ КАНАЛОВ. Если с помощью EXCPW, то сначала в ТАБЛИЦУ ЗАДАЧ записывается условие ожидания той задачи, которая осуществляет запрос на ввод-вывод. После этого управление также передается программе ПЛАНИРОВАНИЕ РАБОТЫ КАНАЛОВ.

ПЛАНИРОВАНИЕ РАБОТЫ КАНАЛОВ является продолжением программ, выполняющих первоначальный анализ. Назначение этой программы — вставить запрос на выполнение операций ввода-вывода в очередь канала, обеспечить его необходимой информацией для инициирования операций ввода-вывода. Функциональная блок-схема программы ПЛАНИРОВАНИЕ РАБОТЫ КАНАЛОВ приведена на рис. 13.6.

Программа ПЛАНИРОВАНИЕ РАБОТЫ КАНАЛОВ получает управление от программ, выполняющих первоначальный анализ. Имеются два входа в эту программу. По первому входу программа ПЛАНИРОВАНИЕ РАБОТЫ КАНАЛА получает управление от программ ПЕРВОНАЧАЛЬНЫЙ АНАЛИЗ, обслуживающих запросы на операцию ввода-вывода для медленных устройств и магнитной ленты. По второму входу она получает управление от программ ПЕРВОНАЧАЛЬНЫЙ АНАЛИЗ, обслуживающих запросы на ввод-вывод для устройств с прямым доступом.

Получив управление, программа ПЛАНИРОВАНИЕ РАБОТЫ КАНАЛОВ анализирует, в какую очередь (главную или альтернативную) надо поставить пришедший запрос. Если главная очередь свободна или альтернативная очередь уже обслуживается, то пришедший запрос размещается в главной очереди, и управление передается программе ДИСПЕТЧЕР. Если главная очередь занята, а альтернативная свободна, то запрос помещается в альтернативную

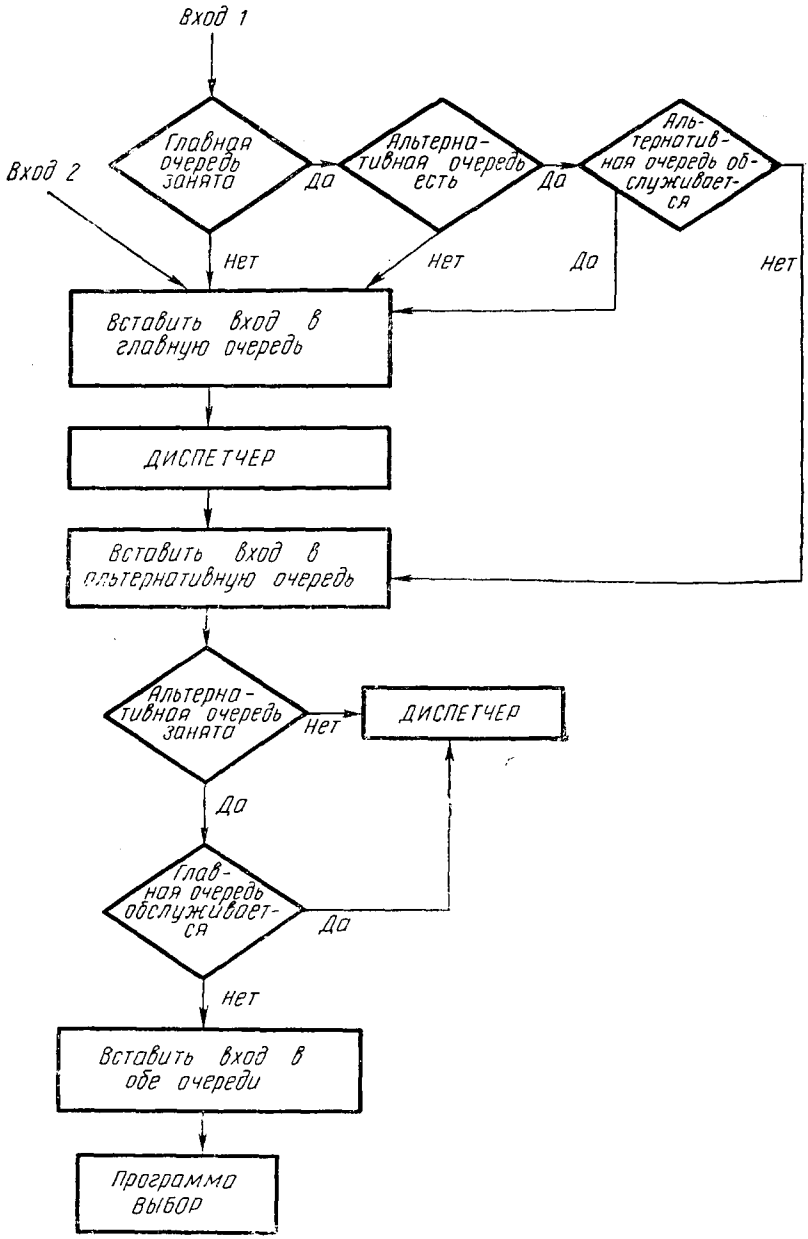


Рис. 13.6. Планирование работы каналов

очередь, и управление также передается программе ДИСПЕТЧЕР. Если и главная, и альтернативная очереди заняты, то запрос на операцию ввода-вывода помещается в обе очереди, и управление передается программе ВЫБОР.

Такая организация очередей позволяет без задержки обслуживать запросы на операцию ввода-вывода. Размещение запроса в обе очереди означает: как только какой-либо из каналов освободится, он сразу же начнет обслуживать ожидающий запрос. При этом этот же запрос вычеркивается из второй очереди.

13.4. Диспетчер

ДИСПЕТЧЕР представляет собой группу программ *Р3*, обслуживающих входы из очередей каналов. Назначение ДИСПЕТЧЕРА: выполнить инициирование операции ввода-вывода; проверить, как прошел запуск операции; установить соответствующие флаги в БЛОКЕ УПРАВЛЕНИЯ ДАННЫМИ, очереди канала, из которой поступил вход на обслуживание, в блоке управления файлами, если обслуживаемый файл находится на устройстве с прямым доступом.

ДИСПЕТЧЕР получает управление в трех случаях: всегда, когда ПЛАНИРОВЩИК КАНАЛОВ размещает вход на запрос операции ввода-вывода в очередь канала; всегда, когда программы окончания ввода-вывода хотят вычеркнуть вход из очереди канала и переупорядочить оставшиеся входы; по специальным запросам СУПЕРВИЗОРа, предназначенным в основном для обнаружения ошибок.

Основные функции ДИСПЕТЧЕРА следующие. Он подготавливает команду запуска устройства для первого входа из очереди канала. Для этого он по соответствующим таблицам определяет цепочку команд канала (для медленных устройств смотри схему взаимодействия таблиц, приведенную на рис. 13.3), помещает адрес слова команды канала из очереди канала в байты 72—75 постоянно фиксируемой области основной памяти и выполняет команду НАЧАТЬ ВВОД-ВЫВОД. Схема инициирования операции ввода-вывода приведена на рис. 13.7.

После инициации операции ввода-вывода производится анализ на основании слова состояния канала, успешно ли осуществлен запуск устройства, и формируются соответствующие флаги в блоке управления данными и блоке управления файлами. Если операция ввода-вывода началась успешно, очередь маркируется как занятая и управление передается программе ВЫБОР. Если инициирование ввода-вывода было неуспешным, то анализируется причина. С этой целью вызываются программы *Р2* обработки ошибок. Вызов программ *Р2* осуществляется выполнением команды обращения к СУПЕРВИЗОРу. Действия этих программ заключаются в следующем. Если причина неуспешного выполнения операции определена, вызываются программы *Р2* обработки распознанных ошибок.

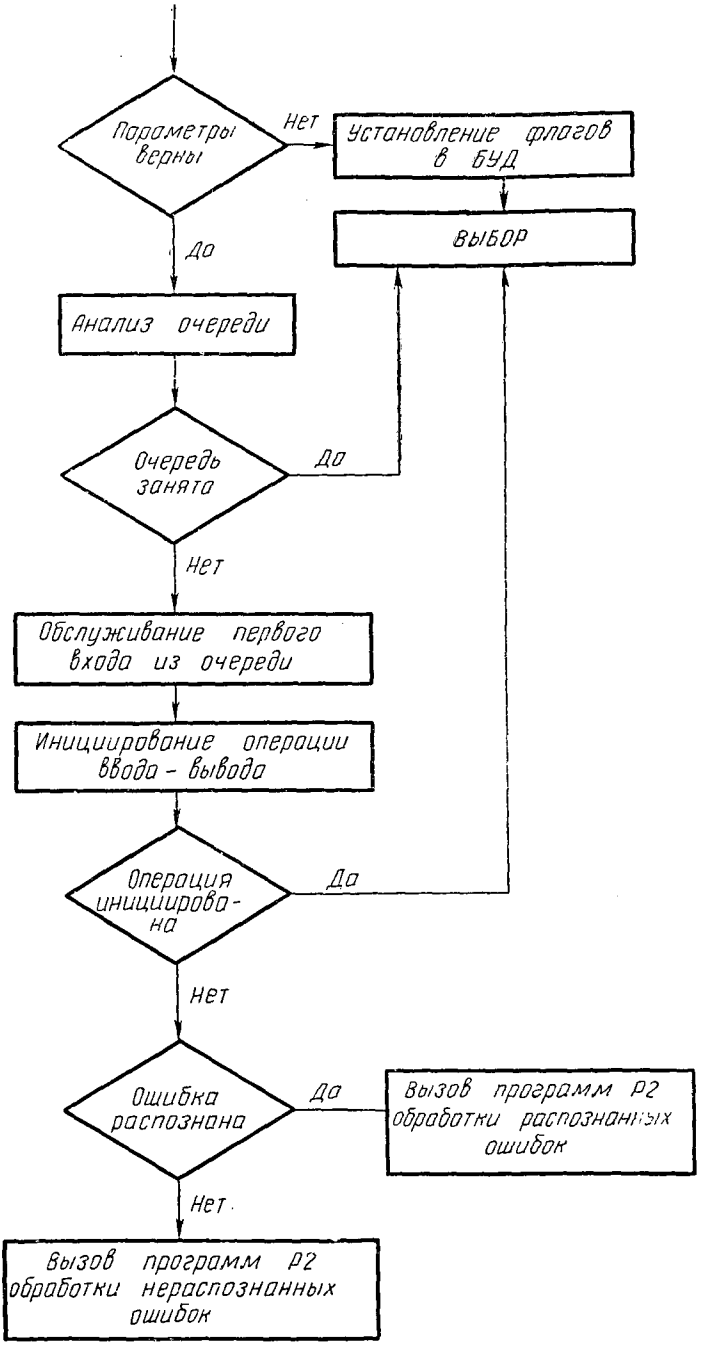


Рис. 13.7. Схема инициирования операции ввода-вывода

Распознанные ошибки обозначают, что появились условия, определяющие невозможность проведения операции ввода-вывода в данный момент. Такие ситуации возникают, когда устройство занято, а канал свободен (например, перематывается магнитная лента). Если канал свободен, то предпринимается попытка найти в очереди канала другое свободное устройство, которое можно инициировать. Запрос, для которого инициирование устройства прошло неуспешно, помещается в конец очереди канала, и обслуживается следующий вход очереди. Действия повторяются до тех пор, пока не будет найдено свободное устройство или вход, для которого первой закончилась неуспешно инициация устройства, не станет снова во главе очереди. При обработке таких ситуаций в заголовке очереди устанавливаются соответствующие флаги, указывающие, что входы очереди ждут обслуживания.

ДИСПЕТЧЕР передает управление программе ВЫБОР. Помеченные флагом ожидания входы будут рассматриваться повторно при обращении к диспетчерским программам в результате следующих действий:

при появлении дополнительного требования на ввод-вывод по этому каналу (новый вход должен быть помещен в очередь);

при появлении прерывания по часам, которое возникает каждые 10 сек. В этом случае для входа, в котором установлен флаг ожидания, выполняется команда обращения к СУПЕРВИЗОРу с требованием выполнить для данного входа программу ДИСПЕТЧЕР;

при появлении команды оператора с пульта. Команда оператора также вырабатывает команду обращения к СУПЕРВИЗОРу с требованием выполнить программу ДИСПЕТЧЕР.

Если причина неуспешного запуска операции ввода-вывода не определена, то вызываются программы *P2* обработки нераспознанных ошибок.

Нераспознанные ошибки появляются в случаях, когда неправильно построена цепочка команд КАНАЛА или адреса данных выходят за пределы основной памяти, или предпринята попытка обратиться к основной памяти, ключ защиты которой отличается от ключа задачи, запросившей ввод-вывод.

Прежде чем вызвать программы распознавания ошибок, ДИСПЕТЧЕР прекращает выполнение операции ввода-вывода, а также устанавливает флаги и записывает соответствующую информацию в БЛОК УПРАВЛЕНИЯ ДАННЫМИ.

Программы обработки нераспознанных ошибок выполняют одно из следующих действий:

если задача, для которой выполнялась операция ввода-вывода, допускает нераспознанную ошибку, то управление передается следующей команде подчиненной программы;

если задача не допускает нераспознанную ошибку, но в подчиненной программе заблаговременно была выполнена макрокоманда STXIT, указывающая СУПЕРВИЗОРу, что в самой проблемной программе есть подпрограмма обработки некоторых типов сбоев, то

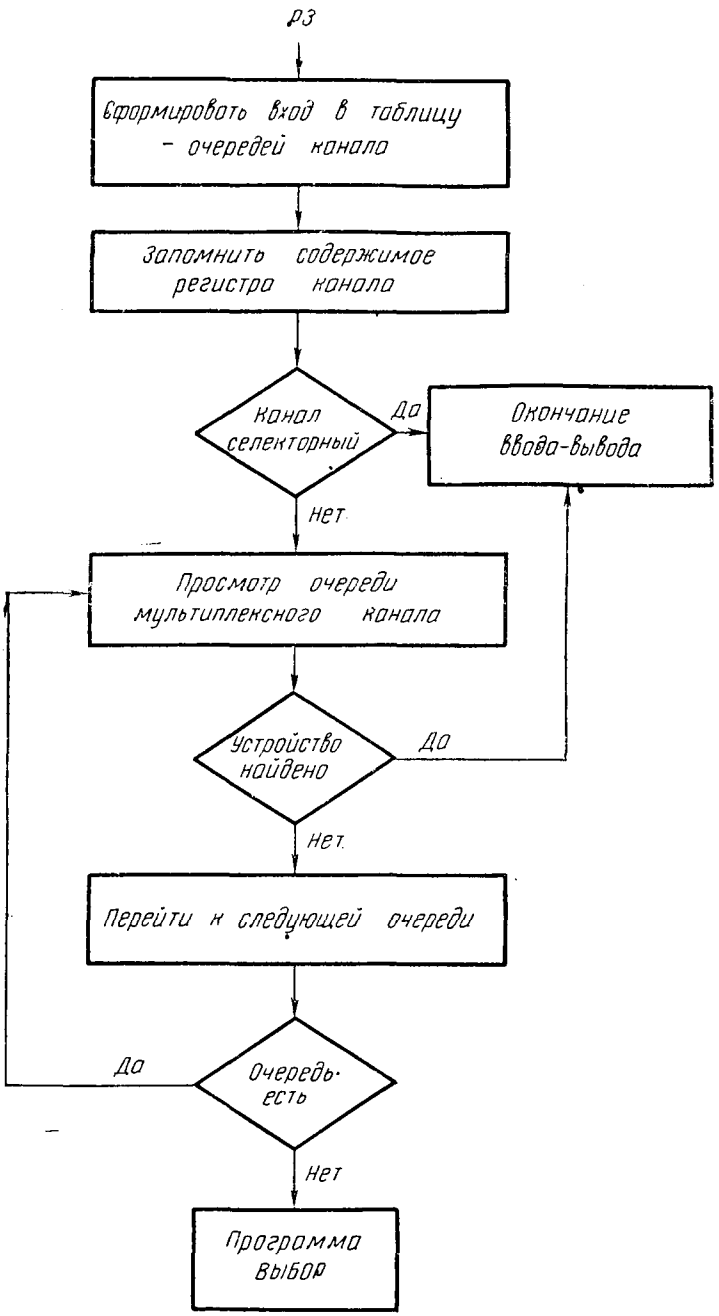


Рис. 13.8. Определение устройства

управление передается этой подпрограмме. Адрес ее находится в ОБЛАСТИ ПАМЯТИ СУПЕРВИЗОРа, принадлежащей проблемной задаче. Действия СУПЕРВИЗОРа по обработке макрокоманды STXIT были описаны в 11.2.

13.5. Программа окончания ввода-вывода

По окончании операции ввода-вывода возникает прерывание от ввода-вывода. Программа АНАЛИЗ ПРЕРЫВАНИЙ передает управление на основании веса прерывания программам *Р3* ОКОНЧАНИЕ ВВОДА-ВЫВОДА. Основное назначение этих программ — проверить, успешно или аварийно закончилась операция ввода-вывода и выполнить действия, связанные с тем или другим окончанием.

Первой работает программа *Р3*, которая определяет устройство, закончившее операцию ввода-вывода. Для этой цели используется таблица очередей канала. Таблица очередей канала содержит столько входов, сколько имеется заголовков очередей. Для каждого заголовка указывается адрес этого заголовка, если он определяет очередь к селекторному каналу. Для заголовков к мультиплексному каналу, помимо адреса этого заголовка, указывается последний адрес, занятый очередями к каналу. Этот адрес предназначен для определения конца очередей.

Блок-схема программы определения устройства, завершившего ввод-вывод, приведена на рис. 13.8.

Вход в программу определения устройства осуществляется из программы *Р3* АНАЛИЗ ПРЕРЫВАНИЙ. На основании веса прерывания формируется вход в ТАБЛИЦУ ОЧЕРЕДЕЙ КАНАЛА.

По таблице определяется, прерывание произошло от селекторного или мультиплексного канала. Если в таблице указан один адрес, то канал (селекторный) и устройство уже определены, так как к селекторному каналу может быть подсоединено только одно устройство. Поэтому управление передается программе ОКОНЧАНИЕ ВВОДА-ВЫВОДА.

Если прерывание произошло от мультиплексного канала, то необходимо просмотреть все его очереди, так как мультиплексный канал имеет столько очередей, сколько устройств к нему подсоединено. Если устройство найдено, то управление также передается программе ОКОНЧАНИЕ ВВОДА-ВЫВОДА. Если устройство не найдено, т. е. при просмотре очередей мультиплексного канала мы дошли до последнего адреса его очередей, управление передается программе ВЫБОР.

Блок-схема программы ОКОНЧАНИЕ ВВОДА-ВЫВОДА приведена на рис. 13.9.

Получив управление от программы определения устройства, программа ОКОНЧАНИЕ ВВОДА-ВЫВОДА выбирает адрес ТАБЛИЦЫ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ из ТАБЛИЦЫ АДРЕСОВ. По ТАБЛИЦЕ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ ОКОНЧАНИЕ ВВОДА-ВЫВОДА определяет вход и главную или альтер-

нативную очередь. Если найденное устройство не содержится ни в одном из входов, то управление передается программе ВЫБОР. Если вход найден, то анализируются флаги заголовка канала, что-

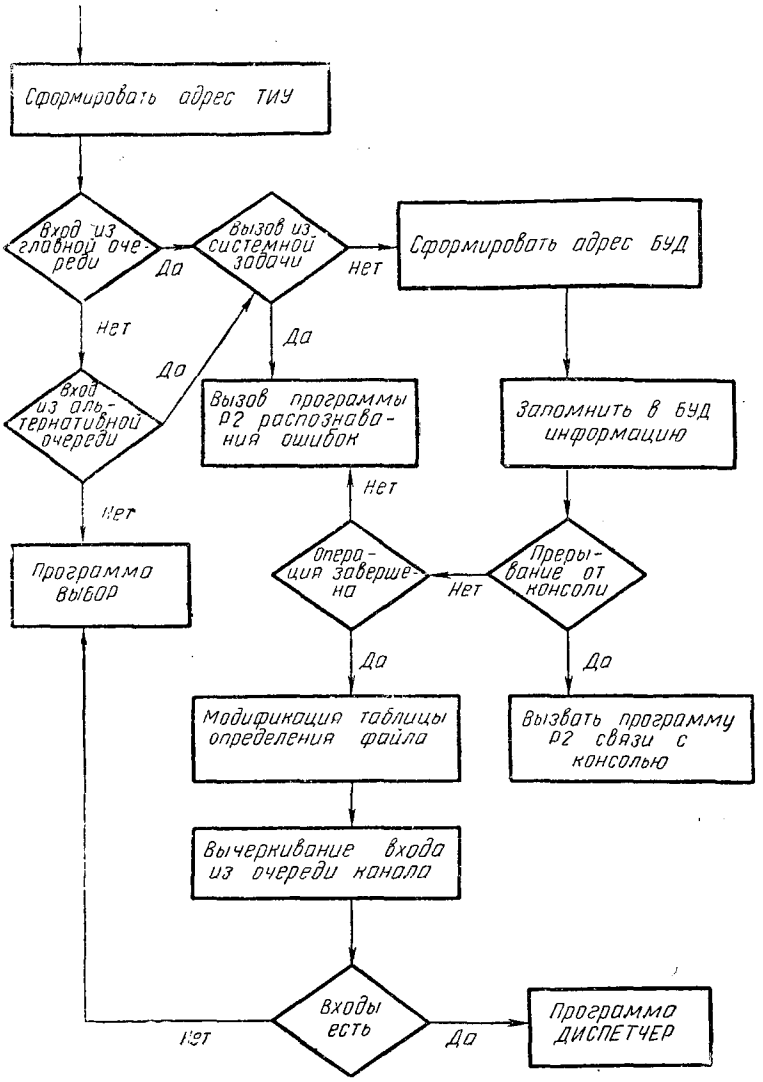


Рис. 13.9. Окончание ввода-вывода

бы определить, закончившаяся операция ввода-вывода была вызвана повторно программами *Р2* распознавания ошибок, когда первый запуск устройств прошел неуспешно, или проблемной программой. Если вызов был осуществлен программами *Р2* распознавания оши-

бок, то управление возвращается им для дальнейшего анализа. Если операция ввода-вывода выполнялась по запросу проблемной задачи, то формируется адрес блока управления данными для ее программы и туда записывается информация из регистра канала. Затем проверяется, является ли обрабатываемое прерывание прерыванием с консоли оператора. Если это так, то вызывается программа *Р2* связи с консолью посредством выполнения макрокоманды обращения к СУПЕРВИЗОРу.

Обращение к программе ОКОНЧАНИЕ ВВОДА-ВЫВОДА производится в случае нормального окончания операции ввода-вывода и в случае аварийного окончания. Поэтому осуществляется проверка, успешно или аварийно закончилась операция ввода-вывода. Если операция закончилась аварийно, то управление передается программам *Р2* распознавания ошибок. При этом, если они распознали ошибку, устройство маркируется как занятое и предпринимается попытка повторить операцию ввода-вывода. Если ошибок не обнаружено, то операция считается завершенной успешно. Информация об этом заносится в блок управления данными, очищается поле Б блока управления, задачи, для которой выполнялась операция ввода-вывода. Последнее означает, что задача готова к выполнению.

Если операция ввода-вывода производилась на устройстве с прямым доступом, то дополнительно проверяется флаг монопольного управления. Если ввод-вывод закончился нормально, то все задачи, ожидающие доступа к такому файлу, также становятся неожидающими.

После того как модифицированы все необходимые таблицы, вход вычеркивается из очереди канала, переопределяется флаг занятости очереди канала, проверяется, есть ли еще входы в этой очереди. Если есть входы в очереди, то управление передается ДИСПЕТЧЕРУ, который начинает обслуживать первый вход очереди. Если очередь обслужена, т. е. входов больше нет, то управление передается программе ВЫБОР.

Мы рассмотрели основные программы *Р3* физической системы управления вводом-выводом, которые раскрывают картину, каким образом происходит обслуживание запросов на операцию ввода-вывода, в каком порядке обслуживаются запросы, как происходит выполнение самих операций ввода-вывода.

ВОПРОСЫ К ГЛАВЕ

1. В какой области памяти находится ТАБЛИЦА ОПРЕДЕЛЕНИЯ ФАЙЛА?
2. Назначение ТАБЛИЦЫ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ.
3. Сколько очередей формируется к одному селекторному каналу, одному мультиплексному каналу?
4. Для каких файлов формируется БЛОК УПРАВЛЕНИЯ ФАЙЛАМИ?
5. При появлении прерывания на запрос операции ввода-вывода какой группе программ передает управление АНАЛИЗ ПРЕРЫВАНИЙ?
6. Какая группа программ получает управление при появлении прерываний по окончании операции ввода-вывода?

7. Какая группа программ выполняет операцию ввода-вывода?
8. Расскажите механизм вызова программ *Р2* распознавания ошибок.
9. Формируется ли задача для вызова программ *Р3*?
10. Что означает «право монопольного управления»?
11. На рис. 13.9 есть несколько «висячих» блоков. В каком случае считается, что программа закончила свои действия?

Глава 14

ПРОГРАММЫ Р2 СУПЕРВИЗОРа

14.1. Назначение программ Р2

Мы уже знаем, что с функциональной точки зрения СУПЕРВИЗОР подразделяется на СУПЕРВИЗОР ЗАДАЧ и СУПЕРВИЗОР ВВОДА-ВЫВОДА. С точки зрения выполнения своих функций и СУПЕРВИЗОР ЗАДАЧ, и СУПЕРВИЗОР ВВОДА-ВЫВОДА состоит из программ *Р2* и *Р3*.

Программы *Р2* обеспечивают разнообразные функции, такие, как связь с оператором, распределение и перераспределение магнитных лент, распознавание и обработка ошибок устройств, загрузка программ в основную память и др. Часть выполняемых ими функций входит в обязательный набор программ СУПЕРВИЗОРа, другие являются дополнительными и могут включаться по требованию пользователя в момент генерации операционной системы. Структура СУПЕРВИЗОРа и метод генерации операционной системы позволяют изменять имеющиеся программы *Р2* или добавлять новые. Причем новые программы могут быть резидентными и транзитными. Пользователь может частично или полностью изменить стандартный набор программ *Р2*, если СУПЕРВИЗОР не выполняет требуемые функции. Однако не только изменения стандартного набора программ *Р2*, но даже чтение отдельных программ должно производиться с крайней осторожностью и значительным знанием структуры СУПЕРВИЗОРа. Прежде всего надо убедиться в полном понимании работы с таблицами СУПЕРВИЗОРа и взаимосвязей отдельных программ.

Рассмотрим общие принципы построения и использования программ *Р2*. Прежде всего следует четко понимать механизм управления программами *Р2*. Как только программы *Р3* определяют функцию для обслуживания анализируемого прерывания, они определяют номер программы *Р2*, которая выполняет эту функцию. Затем необходимо сформировать системную задачу, которой подчиняется данная программа *Р2*. Каждой резидентной программе *Р2* соответствует свой номер задачи, в то время как группе транзитных программ *Р2* может соответствовать один и тот же номер задачи. Номер задачи определяет приоритет системной задачи. Номер программы *Р2* является номером ее загрузочного модуля, который указывается в команде обращения к СУПЕРВИЗОРу. Вызов любой

программы *Р2* осуществляется посредством выполнения команды обращения к СУПЕРВИЗОРу, которая всегда вызывает прерывание.

Формирование системной задачи еще не означает, что программа *Р2* может получить управление. Она получает управление только в том случае, если ее вызвала какая-либо другая задача. Таким образом, программа *Р2* получает управление, если для нее сформирована задача, готовая к выполнению, эту задачу вызвала какая-либо другая задача (проблемная или системная) и нет задач с более высоким приоритетом, готовых к выполнению.

Управление программе *Р2* всегда передает программа ВЫБОР. По окончании действий программа *Р2* должна передать управление программе ВОЗВРАТ посредством выполнения команды обращения к СУПЕРВИЗОРу.

14.2. Соглашения и ограничения

Все программы *Р2* делятся на резидентные и транзитные. К резидентным программам относятся те, которые выполняют функции, наиболее часто встречающиеся при работе системы. Примерами резидентных программ *Р2* служат ЗАГРУЗЧИК, некоторые программы, реализующие операции ввода-вывода, программы обработки конца задания.

Часть программ *Р2* может быть в конкретной операционной системе либо резидентной, либо транзитной в зависимости от того, как определил их пользователь во время генерации системы. Поэтому при проектировании программ *Р2*, как правило, не делается большого различия между резидентными и транзитными программами, и основные соглашения справедливы для всех программ *Р2*.

1. Необходимо ограничить максимальный размер программ *Р2*. Это объясняется тем, что программы *Р2* являются самой подвижной частью СУПЕРВИЗОРа, и их набор может меняться. Кроме того, программы *Р2* могут быть как транзитными, так и резидентными. Длина программ не должна превышать области перекрытия, установленной при проектировании операционной системы для одной программы *Р2*.

2. Местоположение программ *Р2* в основной памяти является фиксированным, так как во время их загрузки в основную память не производится вычисление их адресов.

3. Программы *Р2* являются привилегированными. Это значит, что для них должен быть установлен бит привилегированности. Например, для Системы-4 таким битом является бит 15 регистра состояния прерывания, для системы ИБМ/360 — бит 15 слова состояния программы.

4. Программы *Р2* относятся к классу полностью прерываемых, однако имеется возможность сделать программу непрерываемой. Для этого необходимо изменить маску прерывания с помощью специальной макрокоманды. Подобные действия никогда не должны

препятствовать прерываниям по машинным сбоям. Поэтому, если программа сделана непрерываемой, не следует замаскировывать прерывания от схем контроля машины.

5. Когда происходит прерывание программ *Р2*, сохраняется только содержание регистров 7—15. Содержимое регистров 0—6 не запоминается. Поэтому использование последних в программах *Р2* нежелательно.

6. Все программы *Р2*, закончившие свое выполнение, передают управление программе ВОЗВРАТ посредством выполнения команды обращения к СУПЕРВИЗОРу. Новые программы *Р2* не должны нарушать этого положения.

7. При построении программы *Р2*, выполняющей определенную функцию, всегда следует проанализировать, должна ли эта функция быть выполнена за непрерываемое время. Если это необходимо, то имеет смысл создать не одну, а несколько программ *Р2* для выполнения данной функции, каждая из которых будет выполняться за непрерываемое время.

8. Приоритет программ *Р2* должен быть всегда выше приоритета программ *Р1*.

9. Обращение к программам *Р2* можно осуществить двумя путями:

1) с помощью команды оператора с консоли. После того как оператор набрал на консоли код команды, происходит прерывание, которое анализируется обычным образом программами *Р3*. Для обработки этого прерывания вызывается системная задача, которой подчинена резидентная программа СВЯЗЬ С КОНСОЛЬЮ. Эта программа перерабатывает код команды оператора в номер требуемой программы *Р2*, с помощью команды обращения к СУПЕРВИЗОРу требует создать системную задачу, которой подчинена определенная программа *Р2*, и передать этой задаче управление;

2) с помощью команды обращения к СУПЕРВИЗОРу. Любая программа *Р3* и любая системная задача могут потребовать выполнения какой-либо программы *Р2* посредством выполнения команды обращения к СУПЕРВИЗОРу, указав в качестве параметров номер требуемой программы *Р2*. Управление программе будет передано обычным образом посредством формирования задачи.

14.3. Обеспечение параметрами

В основу создания программ *Р2* положен параметрический принцип. Это значит, что многие действия этих программ определяются на основании параметров, сопровождающих вызов программы *Р2*. Главным способом вызова программ *Р2* является выполнение либо непосредственно команды обращения к СУПЕРВИЗОРу, либо макрокоманды, макрорасширение которой содержит команду обращения к СУПЕРВИЗОРу. Поэтому необходимо рассмотреть соглашения, связанные с понятием макрорасширение. В общем случае вызов любой программы *Р2* представляет собой

команду обращения к СУПЕРВИЗОРу с номером SVC, равным номеру этой программы, и следующими за командой параметрами. Например, макрорасширение макрокоманды WAIT (ЖДАТЬ) на языке АССЕМБЛЕРа будет иметь вид:

```
SVC 54
DC  A(имя файла),
```

где 54 является номером *P2* программы, которая вызывается макрокомандой WAIT, а параметр представляет собой адресную константу.

Чтобы понять, как программа *P2* пользуется своими параметрами, вспомним, что появление SVC вызывает прерывание для обращения к СУПЕРВИЗОРу. Значение Р-счетчика проблемной программы становится параметром к программам обработки прерываний, которые заполняют его в ТАБЛИЦЕ ЗАДАЧ в поле А блока управления прерванной задачи. Одновременно номер вызываемой задачи запоминается в поле В блока управления вызывающей задачи, а номер вызывающей задачи разместится в поле Е блока управления вызываемой задачи.

Таблица 14.1

Мнемоника операции	Операнды	Пояснения
LH	10,84	Адрес таблицы адресов загружается в регистр 10
LH	11,10(10)	Адрес SYSCOS загружается в регистр 11
SLL	11,4	Вычисляется адрес номера точки входа в ТАБЛИЦУ ЗАДАЧ по формуле: $N \times 16 + A$, где A — адрес начала ТАБЛИЦЫ ЗАДАЧ (хранится в таблице адресов); N — номер ЗАДАЧИ, находящийся в SYSCOS; 16 — длина блока управления. Адрес размещается в регистре 11
AH	11,6(10)	
SR	12,12	Очистка регистра 12
IC	12,12(11)	Определение номера вызывающей задачи и размещение его в регистре 12
SLL	12,4	Вычисляется адрес блока управления вызывающей задачи по формуле: $N \times 16 + A$
AH	12,6(10)	
L	13,0(12)	Адрес размещается в регистре 12
		Адрес, где хранится Р-счетчик, пересылается в регистр 13
L	14,0(13)	Первый параметр пересылается в регистр 14
	. . .	Аналогичные действия для выделения следующих параметров
AH	13,=H',4'	Увеличение Р-счетчика на адрес последнего параметра и размещение его в ТАБЛИЦЕ ЗАДАЧ
ST	13,0(12)	

После того как программа СУПЕРВИЗОРа ВЫБОР определит задачу, которой необходимо передать управление, она размещает номер выбранной задачи в ТАБЛИЦУ АДРЕСОВ.

Для того чтобы получить доступ к параметрам, программа *Р2* должна выполнять ряд действий, которые в нашем примере для макрокоманды WAIT (ЖДАТЬ) будут записаны на АССЕМБЛЕРе в виде программы, приведенной в табл. 14.1.

После того как параметры стали доступны вызванной задаче, необходимо самым тщательным образом проверить их корректность. Исключения составляют адресные константы.

В качестве примера возьмем макрокоманду FCOM и посмотрим, что будет, если параметры окажутся неправильными. Макрокоманда FCOM служит для обращения к ОБЛАСТИ ПАМЯТИ СУПЕРВИЗОРа (табл. 15.2), принадлежащей области проблемной задачи.

Задав в качестве параметров относительный адрес (от начала требуемой области) и длину поля, мы можем получить доступ к соответствующему элементу ОБЛАСТИ ПАМЯТИ СУПЕРВИЗОРа. Если адрес требуемой области искажен, то макрокоманда FCOM может записать информацию в область другой задачи или даже в область самого СУПЕРВИЗОРа. Поэтому следует заранее убедиться в корректности параметров.

Корректности параметров можно добиться, проверив, находится ли вычисленный адрес в границах области памяти, отведенной задаче. Верхняя и нижняя границы памяти, отведенной задаче, указаны в ОБЛАСТИ СВЯЗИ СУПЕРВИЗОРа (табл. 15.3). Кроме этого, надо убедиться в том, что адрес выровнен.

Если программа *Р2* использует привилегированные команды, то в листинге программы эти команды должны быть отмечены специальным флагом *Р*. Другого способа убедиться в наличии привилегированных команд нет.

Прежде чем включить подготовленную программу *Р2* в состав СУПЕРВИЗОРа, необходимо выполнить холостой прогон работы системы и убедиться, что последняя функционирует нормально.

ВОПРОСЫ К ГЛАВЕ

1. Назначение программ *Р2*.
2. Как происходит вызов программ *Р2*?
3. Можно ли из программы *Р3* передать управление программе *Р2* посредством команды управления программами?
4. Закреплены ли адреса программ *Р2* при вызове их в основную память?
5. Прерываемы ли программы *Р2*?
6. Какая программа передает управление программам *Р2*?
7. Какой программе должны передать управление программы *Р2* и *Р3*, закончившие свои действия?
8. Какие пути вызова программ *Р2* существуют?
9. Какой принцип положен в основу проектирования программ *Р2*?

Глава 15

УПРАВЛЕНИЕ ЗАДАНИЯМИ

15.1. Управляющие таблицы

УПРАВЛЕНИЕ ЗАДАНИЯМИ является частью управляющей программы, которая осуществляет прием входного потока пакетированных заданий и подготовку системы к выполнению заданиями из очереди принятых заданий. Функции УПРАВЛЕНИЯ ЗАДАНИЯМИ реализуются набором программ, которые выполняются в состоянии процессора *P*1 и *P*2. Те программы, которые выполняются в состоянии процессора *P*2, относятся к программам *P*2 СУПЕРВИЗОРа и не являются собственными программами УПРАВЛЕНИЯ ЗАДАНИЯМИ. Программы, принадлежащие УПРАВЛЕНИЮ ЗАДАНИЯМИ, выполняются в состоянии процессора *P*1, т. е. являются обрабатывающими программами. Однако их приоритет всегда выше приоритета проблемных программ *P*1.

С функциональной точки зрения можно выделить четыре основных программы УПРАВЛЕНИЯ ЗАДАНИЯМИ: ВВОД ЗАДАНИЙ, ПЛАНИРОВЩИК, ГЛАВНЫЙ ПЛАНИРОВЩИК (иногда в литературе его называют РАСПРЕДЕЛИТЕЛЕМ), ПЕРЕРАСПРЕДЕЛИТЕЛЬ.

Как и программы СУПЕРВИЗОРа, программы УПРАВЛЕНИЯ ЗАДАНИЯМИ выполняют модификацию и формирование управляющих таблиц СУПЕРВИЗОРа: ТАБЛИЦЫ ЗАДАЧ, таблицы информации об устройствах, списка распределенной памяти, области связи с СУПЕРВИЗОРом и таблицы УПРАВЛЕНИЯ ЗАДАНИЯМИ, из которых основными являются таблица потоков и область памяти СУПЕРВИЗОРа. С назначением и структурой ТАБЛИЦЫ ЗАДАЧ и таблицы информации об устройствах мы познакомились в § 12.1 и § 13.2. Рассмотрим назначение и структуру остальных таблиц.

СПИСОК РАСПРЕДЕЛЕННОЙ ПАМЯТИ относится к области программ *P*3 и является одной из управляющих таблиц СУПЕРВИЗОРа. Структура списка приведена на рис. 15.1.

Вход каждой задачи содержит начальный и конечный адреса основной памяти, отведенной задаче, которая формируется из шага задания, а также номер потока, через который производился прием задания на выполнение. Число таких входов всегда фиксировано и равно числу задач, которые могут выполняться одновременно.

После входов задач следует таблица адресов, содержащая начальные адреса потоков. Длина таблицы также фиксирована и равна максимальному числу потоков в системе.

Защита памяти производится на основании ключа защиты памяти. Существует шестнадцать ключей. Ключи 0 и 15 предназначены для защиты программ операционной системы. Остальные ключи распределяются между проблемными задачами. Список свобод-

ных ключей содержит маркеры свободных и занятых ключей. Ключ считается занятым, если он присвоен какой-либо задаче, и свободным, если он еще не используется. Выделенный задаче ключ защиты хранится в ее блоке управления в ТАБЛИЦЕ ЗАДАЧ. Далее следует таблица разделов. Она содержит по одному входу для каждого раздела. Вход одного раздела содержит информацию, какие ключи защиты памяти, приоритеты задач, номера потоков будут выделены задачам, выполняемым в этом разделе.

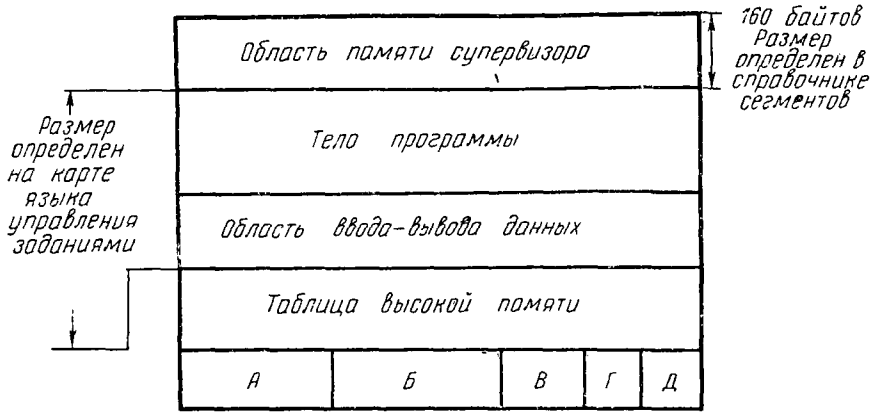


Рис. 15.1. Структура списка распределенной памяти

Для каждого потока создается таблица, структура которой приведена в табл. 15.1. В таблицу потока помещается информация об отведенных этому потоку устройствах. С этой целью в таблице перечисляются номера входов в таблицу информации об устройствах для выделенных потоку устройств. Кроме того, в таблице хранится ключ защиты памяти, который связан с этим потоком, и номер точки входа в ТАБЛИЦУ ЗАДАЧ блока управления задачей, который соответствует программе ВВОД ЗАДАНИЙ. Это делается для ускорения вызова программы ВВОД ЗАДАНИЙ для приема заданий из данного потока.

Таблица 15.1

Байты	Содержание
0—5	Резервируется
6—7	Номера входов в таблицу информации об устройствах для устройств с прямым доступом
8—9	Номера входов в таблицу информации об устройствах для медленных устройств
10	Ключ защиты
11	Номер задачи, соответствующий программе ВВОД ЗАДАНИЙ
12	Резервируется

Таблица 15.2

Байты	Содержимое
0—3	Содержимое регистра состояния прерывания
4—67	Содержимое общих регистров
68—99	Содержимое регистров с плавающей запятой
100—103	Вес прерывания
104—107	Р-счетчик — запоминается в момент входа в программу обработки распознанных ошибок
108—115	Содержимое общих регистров — запоминается в момент входа в программу обработки распознанных ошибок
116—119	Р-счетчик — запоминается в момент входа в программу связи с оператором
120—127	Содержимое регистров — запоминается в момент входа в программу связи с оператором
128—131	Р-счетчик — запоминается в момент входа в программу обработки нераспознанных ошибок
132—139	Содержимое регистров — запоминается в момент входа в программу обработки нераспознанных ошибок
140—143	Резервируются
144—147	Р-счетчик. Запоминается при аварийном переходе на обработку конца задания
148—151	Р-счетчик — запоминается при входе в отладочные программы
152—159	Содержимое — запоминается при входе в отладочные программы

В каждой обрабатывающей программе первые 160 байтов отводятся под область для СУПЕРВИЗОРа. В этой области СУПЕРВИЗОР запоминает значение Р-счетчика и содержимое общих регистров и регистров с плавающей запятой во время прерывания программы для ее восстановления после обработки прерывания. Эта область доступна программам СУПЕРВИЗОРа и проблемной программе. Структура области памяти СУПЕРВИЗОРа приведена в табл. 15.2.

Из этой области выбирается значение Р-счетчика и регистров программой ВЫБОР, когда она выбирает проблемную задачу, чтобы передать ей управление. В байтах 0—3 запоминается содержимое регистра состояния прерывания, поэтому проблемная программа имеет возможность проверить причину прерывания. В байтах 4—99 запоминается содержимое общих регистров и регистров с плавающей запятой. В байтах 100—103 запоминается вес прерывания. На его основе также можно производить анализ прерываний в проблемной программе.

Проблемная программа может иметь собственные подпрограммы обработки ошибок. Связь этих программ с СУПЕРВИЗОРом осуществляется посредством макрокоманды STXIT. Адреса этих программ располагаются в ОБЛАСТИ СВЯЗИ С СУПЕРВИЗОРом, как только первый раз встретится макрокоманда STXIT.

Кроме программ обработки ошибок, проблемная программа может включать подпрограмму связи с оператором через консоль. Такая программа необходима в случаях, когда нормальное выполне-

ние проблемной программы требует постоянного вмешательства оператора. Адрес этой программы также размещается в ОБЛАСТИ СВЯЗИ СУПЕРВИЗОРа при появлении макрокоманды STXIT.

Все эти программы могут использовать только два общих регистра с номерами 10 и 11. Когда СУПЕРВИЗОР передает управление одной из проблемных программ: обработка распознанных ошибок, обработка нераспознанных ошибок, связь с консолью, — он запоминает значение Р-счетчика и содержимое общих регистров с номерами 10 и 11 соответственно в байтах 104—115, 128—139, 116—127. При выходе из этих программ СУПЕРВИЗОР возвратит управление проблемной программе, восстановив значения Р-счетчика и общих регистров на основании информации, хранящейся в соответствующих байтах ОБЛАСТИ ПАМЯТИ СУПЕРВИЗОРа.

При аварийном окончании проблемной программы СУПЕРВИЗОР запоминает значение Р-счетчика в байтах 144—147 и передает управление одной из своих программ обработки конца задания.

Для отладки проблемной программы используются отладочные программы, которые могут использовать только общие регистры с номерами 10 и 11. Поэтому при передаче управления отладочным программам запоминается только содержимое этих регистров и значение Р-счетчика в байтах 148—159.

Таблица 15.3

Байты	Содержимое
0—5	Текущая дата
6—8	Дата создания
9—11	Начальный адрес основной памяти, отведенный задаче
12—22	Область связей
23	Маска программы
24—35	Идентификатор программы
36—39	Адрес таблиц высокой памяти
40—45	Номер прогона
46—47	Число требуемых магнитных лент
48—51	Адрес программы обработки распознанных ошибок в проблемной программе
52—55	Адрес программы связи с оператором в проблемной программе
56—59	Адрес программы обработки нераспознанных ошибок в проблемной программе
60—63	Последний адрес основной памяти, отведенной задаче

ОБЛАСТЬ СВЯЗИ СУПЕРВИЗОРа является одной из таблиц СУПЕРВИЗОРа и хранится в области программ *РЗ*. Ее структура приведена в табл. 15.3. Такая область создается для каждой выполняемой задачи. Начальный адрес первой из этих областей находится в ТАБЛИЦЕ АДРЕСОВ СУПЕРВИЗОРа. Адрес любой области

легко вычисляется. Он равен ключу защиты памяти, отведенному задаче, умноженному на длину области, т. е. 64, плюс начальный адрес этих областей. Ключ защиты памяти хранится в поле И блока управления задачей в ТАБЛИЦЕ ЗАДАЧ.

В байты 0—5 заносится текущая дата выполнения задачи. Эта дата будет указана в распечатке программы, подчиненной данной задаче, в случае ее аварийного окончания.

В байтах 6—8 указывается месяц и год создания программы. Эта дата служит для анализа при модификации программы, чтобы не запутаться в листингах, полученных при распечатке различных версий программы.

В байтах 9—12 и 60—63 хранятся начальный и конечный адреса основной памяти, отведенной задаче. Они используются программами *Р2* СУПЕРВИЗОРа для проверки правильности вычисленных ими адресов внутри программы. Если вычисленный адрес меньше начального или больше конечного, то он не принадлежит данной программе, а следовательно, неправильно заданы параметры в макрокомандах обращения к СУПЕРВИЗОРу.

Байты 12—22 предназначены для записи СУПЕРВИЗОРом информации, которая ему требуется для управления проблемными задачами.

В байте 23 хранится маска программы. С помощью маски программы можно запретить некоторые виды прерывания программы (например, прерывание при возникновении переполнения в операциях с фиксированной запятой).

В байтах 24—35 хранится идентификатор программы, подчиненной данной задаче.

В байтах 36—39 находится адрес таблиц ВЫСОКОЙ ПАМЯТИ. К ним относятся некоторые таблицы СУПЕРВИЗОРа, которые располагаются внутри проблемной программы: таблица параметров файлов, таблица назначения физических устройств логическим устройствам и другая информация.

В байты 40—45 заносятся номера предыдущего и текущего прогона задачи. Они используются макрокомандами ОТКРЫТЬ и ЗАКРЫТЬ (OPEN и CLOSE).

В байтах 46—47 указывается число используемых магнитных лент, которое используется программами управления данными при обработке файлов на магнитных лентах.

В байтах 48—51, 52—55, 56—59 указываются начальные адреса подпрограмм проблемной программы, предназначенных для обработки ошибок и связи с оператором. Эти адреса устанавливаются при обнаружении в проблемной программе макрокоманды STXIT.

Для выполнения любой задачи ей необходимо отвести основную память. Структура этой области памяти для всех проблемных задач одинакова и приведена на рис. 15.2.

Первые 160 байтов области основной памяти, отведенной задаче, представляют собой область памяти СУПЕРВИЗОРа. Область фор-

мируется во время загрузки программы. За ней следует область, в которой располагается тело самой программы. Структура программы может быть любой из допустимых операционной системой структур. Размер этой области хранится в справочнике сегментов и определяется программой РЕДАКТОР во время помещения программы в одну из системных библиотек.

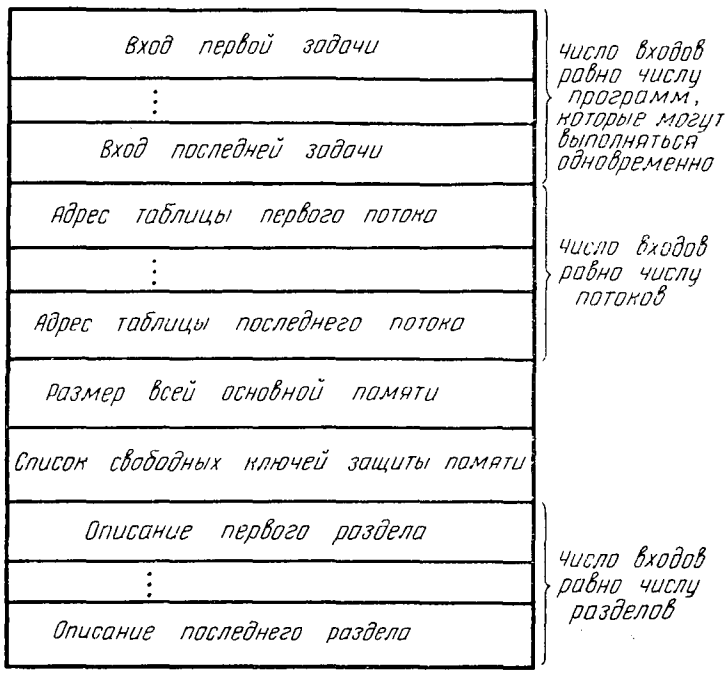


Рис. 15.2. Структура основной памяти, отведенной задаче

Если программа использует динамические файлы данных (файлы, которые не загружаются одновременно с программой, а вводятся или формируются при выполнении программы), то вслед за областью с телом программы отводится область для ввода-вывода данных. Этой области может и не быть, если программа не использует динамические файлы, а вся требуемая для выполнения информация загружается в основную память во время загрузки самой программы. Размер областей, отведенных под тело программы и для ввода-вывода данных, указывается на одной из управляющих карт языка управления заданиями. При отсутствии динамических файлов размер требуемой памяти на карте будет совпадать с размером, хранящимся в справочнике сегментов.

Далее следуют таблицы ВЫСОКОЙ ПАМЯТИ. В этой области хранятся: таблица параметров файлов, которая содержит адреса

таблиц определения файлов; таблиц назначения физических устройств логическим устройствам; таблиц томов на устройствах с прямым доступом; параметры времени прогона. Таблицы формируются программами УПРАВЛЕНИЯ ЗАДАНИЯМИ на основании управляющих карт языка управления заданиями.

Затем следует поле, содержащее пять адресов. Адрес поля указан в ОБЛАСТИ СВЯЗИ СУПЕРВИЗОРа в байтах 36—39. Символом А обозначается последний адрес тела программы. Символом Б — адрес параметров времени прогона, находящихся в высокой памяти. Символом В — адрес таблицы томов на устройствах с прямым доступом. Символом Г — адрес таблицы назначения физических устройств логическим. Символом Д — адрес таблицы параметров файлов.

По адресу, хранимому в ОБЛАСТИ СВЯЗИ СУПЕРВИЗОРа в байтах 36—39, можно найти адрес требуемой таблицы, находящейся внутри области памяти, отведенной задаче, используя поле с адресами. Определив требуемый адрес, можно получить доступ к нужной таблице.

Ключ защиты памяти, отведенной задаче, хранится в ТАБЛИЦЕ ЗАДАЧ в поле И блока управления.

15.2. Ввод заданий

С функциональной точки зрения первой программой, выполнение которой должно предшествовать выполнению других программ УПРАВЛЕНИЯ ЗАДАНИЯМИ, является ВВОД ЗАДАНИЙ. Ее назначение — принять входной поток заданий, проверить управляющие параметры, сформировать очередь заданий потока, через который был осуществлен прием входного потока заданий.

С этой целью программа ВВОД ЗАДАНИЙ выполняет следующие функции:

просматривает все карты, относящиеся к текущему заданию, и анализирует, нет ли в них ошибок;

если необходимо, каталогизирует описания заданий для дальнейшего использования и корректирует каталог заданий;

формирует входы для заданий и размещает их в очередь заданий;

подготавливает информацию, которая будет размещена в управляющих таблицах;

производит начальное формирование таблиц высокой памяти: создает входы в таблицу параметров файлов, таблицу назначения физических устройств логическим устройствам, таблицу параметров времени прогона;

переупорядочивает очереди заданий и корректирует описания катализированных процедур;

размещает исходные данные проблемных задач в одной из библиотек системы;

анализирует, могут ли быть удовлетворены требования на ресур-

сы текущего задания в том потоке, в котором осуществлялся ввод задания.

ВВОД ЗАДАНИЙ можно вызвать для выполнения четырьмя путями: по команде оператора с консоли, из программы РЕДАКТОР, с помощью специальной макрокоманды из проблемной программы, косвенным путем посредством макрокоманд вызова ЗАГРУЗЧИКа из проблемной программы. Во всех случаях вызов означает формирование задачи, соответствующей программе ВВОД ЗАДАНИЙ, и передачу этой задаче управления.

Обычным путем, которым необходимо инициировать систему на ввод и обработку заданий, является вызов программы ВВОД ЗАДАНИЙ с консоли посредством команды оператора. По этой команде сначала инициируется системная задача, которой подчинена программа *Р2* СУПЕРВИЗОРа, проверяющая правильность вызова ВВОДА ЗАДАНИЙ. Она запоминает номер консоли, с которой потребовали ввод заданий, и проверяет, находится ли в основной памяти программа ВВОД ЗАДАНИЙ. Проверка производится по ТАБЛИЦЕ ЗАДАЧ. Если ВВОД ЗАДАНИЙ в памяти не находится, программа *Р2* пытается отвести ей участок в ТАБЛИЦЕ ЗАДАЧ для размещения блока управления задачей. Если свободных участков нет, то программа *Р2* передает управление программе ВОЗВРАТ. Если нужный участок найден в ТАБЛИЦЕ ЗАДАЧ, то программа *Р2* выполняет подготовку для вызова ЗАГРУЗЧИКа, который должен загрузить ВВОД ЗАДАНИЙ в основную память. Одновременно программа *Р2* осуществляет проверку, чтобы убедиться в том, что входной поток заданий будет принят в нужный раздел и этому разделу отведено требуемое число устройств. После этого программа *Р2* вызывает программу *Р1*, которая является составной частью УПРАВЛЕНИЯ ЗАДАНИЯМИ. Вызов программы *Р1* также означает формирование для нее проблемной задачи. Эта программа *Р1* может быть вызвана только из рассмотренной программы *Р2*. Она модифицирует ТАБЛИЦУ ПОТОКОВ в соответствии с информацией, которую для нее подготовила программа *Р2*, а также пересылает сообщения о встреченных ошибках в СИСТЕМНЫЙ ЖУРНАЛ.

Программа *Р1*, как и программа *Р2*, имеет два выхода. Если были обнаружены какие-либо ошибки, то посредством макрокоманды обращения к СУПЕРВИЗОРу она передает управление программам СУПЕРВИЗОРа обработки ошибок. Если все в порядке, то она вызывает ЗАГРУЗЧИК для загрузки программы ВВОД ЗАДАНИЙ посредством специальной макрокоманды.

Вызов программы ВВОД ЗАДАНИЙ можно также осуществить из программы РЕДАКТОР. Это производится в том случае, когда один шаг задания потребовал отредактировать программу, а второй шаг — выполнить отредактированную программу. После выполнения редактирования программы РЕДАКТОР формирует выходной поток СИСТЕМЫ РЕДАКТИРОВАНИЯ ПРОГРАММ, который содержит управляющие операторы описания задания, текст програм-

мы и размещается в основной главе вводных данных. Для того чтобы выполнить отредактированную программу, необходимо ввести сформированное задание на ее выполнение и разместить его в очередь заданий. Ввод заданий осуществляется только программой ВВОД ЗАДАНИЙ. Поэтому РЕДАКТОР вызывает ВВОД ЗАДАНИЙ, который, закончив свои действия, вновь передает управление РЕДАКТОРу.

Из проблемной программы ВВОД ЗАДАНИЙ можно вызвать двумя способами. Первый способ — вызов осуществляется посредством макрокоманды вызова ВВОДА ЗАДАНИЙ. В этом случае ВВОД ЗАДАНИЙ вызывается как транзит самой проблемной программы на ее место. Выполнив свои действия, ВВОД ЗАДАНИЙ возвращает управление проблемной программе, которая его вызвала. Второй способ — посредством макрокоманд вызова ЗАГРУЗЧИКа, указав в качестве параметра имя программы ВВОД ЗАДАНИЙ. В этом случае ВВОД ЗАДАНИЙ также располагается в области проблемной программы, а передает управление программе СУПЕРВИЗОРа окончания задания.

Исходными данными для работы программы ВВОД ЗАДАНИЙ являются управляющие операторы языка управления заданиями, которые находятся во входном потоке заданий. Для разных вычислительных систем обозначение и число управляющих операторов различно, но назначение их одинаковое. Все управляющие операторы делятся на следующие группы в зависимости от их функционального назначения: идентификация задания, информация о программе, описание файлов, подготовка к выполнению задания, управление данными, управляющие операторы, управление входным потоком задания. Рассмотрим назначение каждой группы операторов.

Операторы группы идентификации задания содержат идентификацию двух типов: информацию, на основании которой можно отделить одно задание от другого, и информацию, на основании которой можно определить имя задания.

По имени задания можно всегда найти в описании задания любые характеристики задания. Эта группа управляющих операторов всегда должна присутствовать во входном потоке заданий.

Операторы группы подготовки к выполнению задания определяют имя той программы (точнее, ее корневого сегмента), которая должна быть выполнена в одном шаге задания.

На основании операторов группы информации о программе можно получить сведения о том, где находится программа, выполнение которой потребовано в шаге данного задания (во входном потоке, библиотеке загрузочных модулей, библиотеке исходных модулей, библиотеке объектных модулей, во временной библиотеке); какой первоначальный приоритет должна иметь программа, в каком разделе надо ее выполнить, какой размер основной памяти необходимо ей выделить, какова должна быть маска программы. Управляющие операторы этой группы задают, где искать описание данного задания. Если задание не является каталогизированным, то его описа-

ние должно быть во входном потоке заданий. Если задание каталогизированное, то его описание следует искать в основной главе с библиотекой каталогизированных процедур.

Группа операторов описания файлов задает назначение физических устройств логическим устройствам, используемым в программе данного шага задания. На основании информации, заключенной в этих операторах, УПРАВЛЕНИЕ ЗАДАНИЯМИ выделяет требуемые устройства заданию, модифицирует ТАБЛИЦУ ПОТОКОВ, ТАБЛИЦУ ИНФОРМАЦИИ об УСТРОЙСТВАХ, формирует таблицу определения файлов. Из этих операторов УПРАВЛЕНИЕ ЗАДАНИЯМИ черпает информацию о наличии стандартных и нестандартных меток файлов.

Группа операторов управления данными задает информацию о файлах, используемых программой и расположенных на устройствах с прямым доступом. На ее основе производится модификация блока управления файлами.

Группа управляющих операторов указывает, какие изменения необходимо внести в описание задания. Например, необходимо переименовать программу или все задания, изменить ранг задания, вычеркнуть задание.

Операторы управления входным потоком дают указания программе ВВОД ЗАДАНИЙ временно приостановить свои действия.

Входными данными для программы ВВОД ЗАДАНИЙ являются управляющие параметры операторов описания задания, которые снабжают ее информацией для выполнения требуемых функций. Выходными данными этой программы будут модифицированная таблица потоков, очереди заданий для потока, информация для таблиц высокой памяти, список сообщений об обнаруженных ошибках. Список сообщений об ошибках либо сразу выводится на печатающее устройство, либо записывается в СИСТЕМНЫЙ ЖУРНАЛ для более поздней распечатки. Очереди заданий потоков размещаются в основной главе с библиотекой заданий, а описания заданий и справочник заданий, на основании которого пишется описание задания, помещаются в основную главу с библиотекой каталогизированных процедур.

ВВОД ЗАДАНИЙ — защищенная, но непривилегированная программа *P*1 УПРАВЛЕНИЯ ЗАДАНИЯМИ. Функциональная блок-схема этой программы приведена на рис. 15.3.

Каждый параметр оператора языка управления заданиями занимает одну перфокарту. Входной поток задания вводится с устройства чтения с перфокарт. Однако для некоторых заданий, которые были оформлены в виде каталогизированных процедур, описания могут находиться на системном диске в основной главе с библиотекой каталогизированных процедур. Кроме этого, задания могут формироваться программой РЕДАКТОР и размещаться в основной главе входных данных. Поэтому, прежде чем начать обработку отдельных управляющих операторов, необходимо установить, где рас-

положено описание текущего задания. Управляющие параметры вводятся по одной карте.

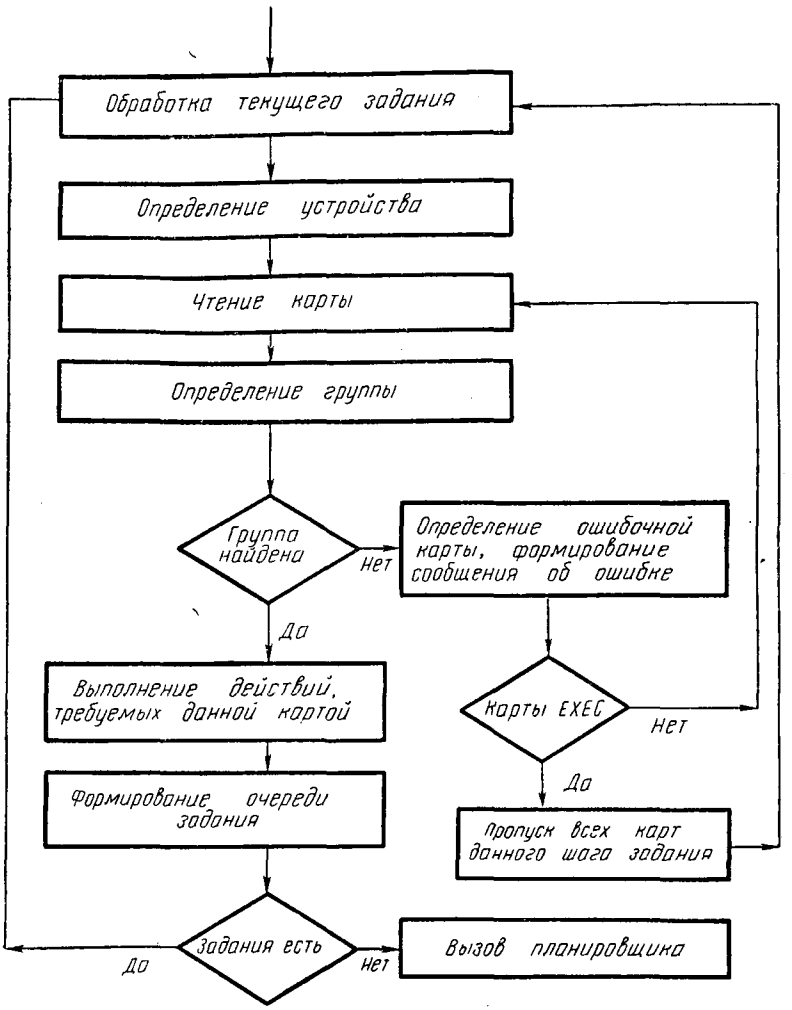


Рис. 15.3. Ввод заданий

Сначала тщательно анализируется правильность управляющих параметров. Проверка идет следующим образом. Проверяется, какой группе принадлежит каждая карта, затем выполняются стандартные проверки данной группы перфокарт. Определяется карта, в которой обнаружена ошибка. Если карта с ошибкой отмечает конец описания задания или определяет параметры времени прогона программы, то данное задание не включается в очередь заданий.

Все проверенные и непроверенные перфокарты пропускаются до тех пор, пока не появится карта, маркирующая начало следующего задания, которое и поступает на обработку. Любая другая ошибочная карта опускается, и управление передается на чтение и проверку следующей карты. В обоих случаях формируется сообщение о найденной ошибке.

Если все карты описания задания правильные, то по переключателю управление передается на обработку каждой карты. Для каждой карты существует своя программа, которая выполняет действия, требуемые этой картой.

Выполнив требуемые действия, модифицировав соответствующие таблицы, подготовив информацию, необходимую для работы других программ УПРАВЛЕНИЯ ЗАДАНИЯМИ, ВВОД ЗАДАНИЙ, формирует вход для данного задания и размещает его в очереди заданий потока, помещает очередь в библиотеку заданий. Описания заданий, которые в дальнейшем будут вызываться с помощью каталогизированной процедуры, в сжатом формате записываются в библиотеку каталогизированных процедур. Туда же помещается справочник заданий, в котором хранятся адреса описаний заданий.

Если обработаны не все задания, то ВВОД ЗАДАНИЙ переходит к обработке следующего задания. Если все задания обработаны, то с помощью макрокоманды обращения к СУПЕРВИЗОРу ВВОД ЗАДАНИЙ вызывает ПЛАНИРОВЩИК.

15.3. Планировщик

ПЛАНИРОВЩИК является программой *P*1 УПРАВЛЕНИЯ ЗАДАНИЯМИ. Входными данными для ПЛАНИРОВЩИКа являются таблицы, подготовленные ВВОДОМ ЗАДАНИЙ, которые содержат ранги заданий, потоки, которым задания принадлежат, список устройств ввода-вывода, требуемых заданию, объем основной памяти. На основании этой информации ПЛАНИРОВЩИК проверяет и модифицирует ТАБЛИЦУ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ, таблицу параметров файлов, которая будет размещена в области памяти проблемной программы, СПИСОК РАСПРЕДЕЛЕННОЙ ПАМЯТИ.

ПЛАНИРОВЩИК выполняет следующие функции:

проверяет, есть ли возможность обеспечить задание требуемым количеством устройств;

проверяет, есть ли в наличии требуемая память;

изменяет СПИСОК РАСПРЕДЕЛЕННОЙ ПАМЯТИ;

проверяет, соответствуют ли требования на устройства их назначению, при необходимости запрашивает специфические устройства. Например, если в наличии нет печатающего устройства, делается запрос, нельзя ли заменить его псевдоустройством, т. е. вывести данные в СИСТЕМНЫЙ ЖУРНАЛ, а когда печатающее устройство освободится, вывести результаты;

читает из библиотеки заданий таблицы высокой памяти, сфор-

мированные программой ВВОД ЗАДАНИЙ, и составляет единые таблицы: упорядочивает таблицы назначения физических устройств логическим;

проверяет, чтобы число требуемых дисков не превышало числа имеющихся в наличии в данной конфигурации машины. Проверяет, имеются ли свободные диски, которые могут быть использованы по требованию проблемных программ.

Вызов программы ПЛАНИРОВЩИКА может быть осуществлен в четырех случаях: по требованию от программ *P*2, которым ПЛАНИРОВЩИК выделен как ресурс системы; по окончании выполнения программы ВВОД ЗАДАНИЙ; по требованию макрокоманды CLOSE (ЗАКРЫТЬ), когда надо перераспределить освободившиеся устройства; по команде оператора с консоли. Во всех случаях сначала вызывается программа *P*2, являющаяся второй фазой ПЕРЕРАСПРЕДЕЛИТЕЛЯ и относящаяся к программам СУПЕРВИЗОРа.

Вторая фаза ПЕРЕРАСПРЕДЕЛИТЕЛЯ может быть вызвана тремя способами: из первой фазы ПЕРЕРАСПРЕДЕЛИТЕЛЯ посредством задания соответствующих параметров в макрокоманде обращения к СУПЕРВИЗОРу для вызова программы СОГЛАСОВАНИЕ; от ВВОДА ЗАДАНИЙ посредством макрокоманды вызова ПЛАНИРОВЩИК; по команде оператора с консоли, которая выполнит, как и в первом случае, команду обращения к СУПЕРВИЗОРу для вызова программы СОГЛАСОВАНИЕ.

Основные функции второй фазы ПЕРЕРАСПРЕДЕЛИТЕЛЯ следующие:

проверка по СПИСКУ РАСПРЕДЕЛЕННОЙ ПАМЯТИ, есть ли свободный ключ, и если есть, помещение его в ТАБЛИЦУ ЗАДАЧ в блок управления ПЛАНИРОВЩИКОМ;

проверка по СПИСКУ РАСПРЕДЕЛЕННОЙ ПАМЯТИ, имеется ли достаточный объем основной памяти для размещения ПЛАНИРОВЩИКА, и если имеется, выделение соответствующей ему задаче основной памяти;

формирование для ПЛАНИРОВЩИКа ОБЛАСТИ СВЯЗИ СУПЕРВИЗОРа;

проверка по таблице приоритетов для программ *P*1, имеет ли ПЛАНИРОВЩИК наивысший приоритет, и если нет, изменение его приоритета на наивысший;

подготовка вызова ЗАГРУЗЧИКа для загрузки ПЛАНИРОВЩИКа в основную память и передачи ему управления.

Как и все программы *P*2, вторая фаза ПЕРЕРАСПРЕДЕЛИТЕЛЯ заканчивает свое выполнение передачей управления программе ВОЗВРАТ.

Из рис. 10.13, который иллюстрирует работу СУПЕРВИЗОРа в мультипрограммном режиме, видно, что первым активизируется раздел с наивысшим приоритетом, который территориально наиболее удален от области управляющей программы. Это значит, что ПЛАНИРОВЩИК должен выбрать задание, которое будет выпол-

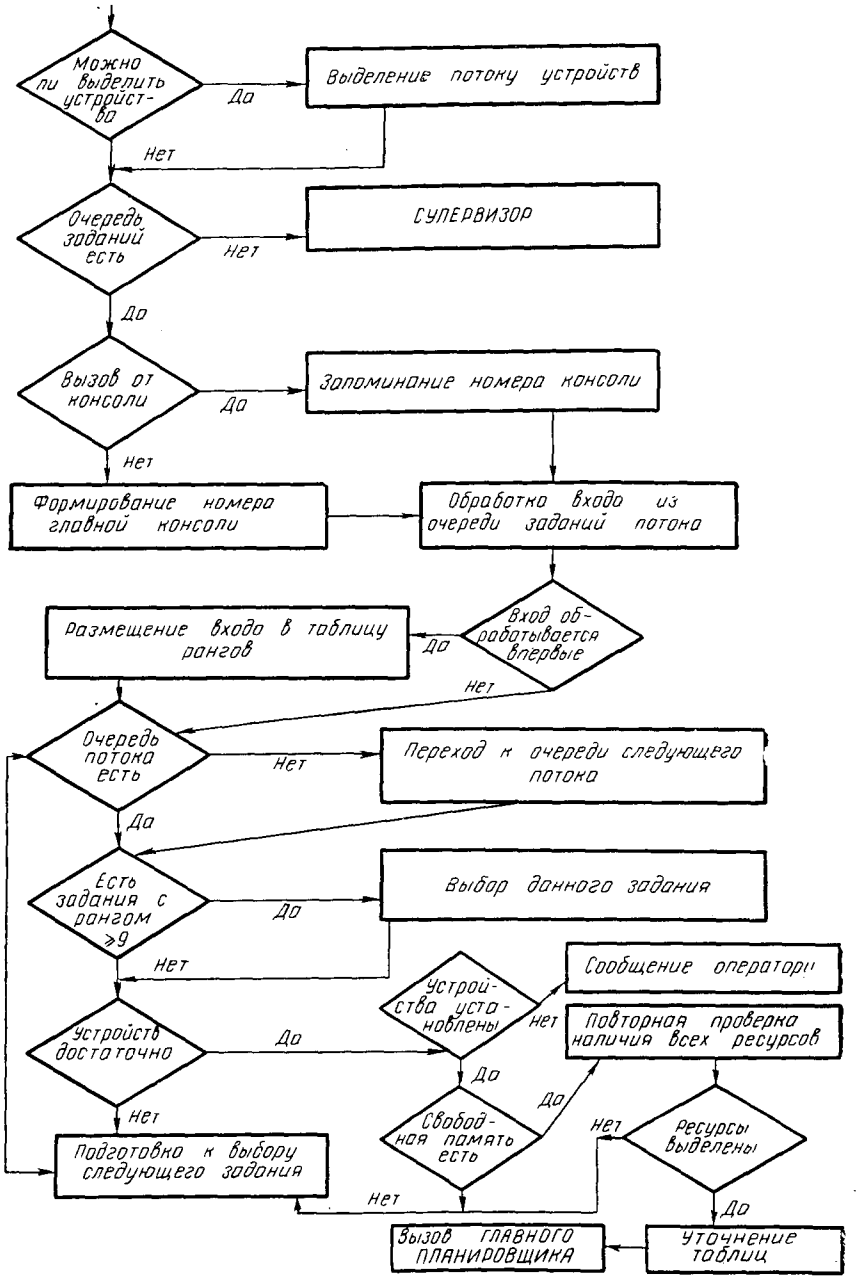


Рис. 15.4. ПЛАНИРОВЩИК

няться, из очереди заданий потока, которому принадлежит этот раздел. Если задание из этого потока можно активизировать, то ему будут выделены ресурсы. Если заданий, которые можно активизировать, в данном потоке нет, то ПЛАНИРОВЩИК переходит к анализу очереди заданий следующего потока. При этом анализ очередей заданий потоков будет производиться в той же последовательности. Функциональная блок-схема программы ПЛАНИРОВЩИК приведена на рис. 15.4.

Получив управление, ПЛАНИРОВЩИК проверяет, есть ли в наличии свободные устройства. Если они есть, то они будут выделены данному потоку для распределения между его заданиями. С этой целью производится модификация ТАБЛИЦЫ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ и ТАБЛИЦЫ ПОТОКОВ. Затем проверяется, есть ли очереди заданий. Если очередей нет, ПЛАНИРОВЩИК заканчивает свою работу, передав управление СУПЕРВИЗОРу. Если очереди заданий есть, то ПЛАНИРОВЩИК анализирует, откуда произошел его вызов. Это необходимо для того, чтобы сформировать номер консоли, который будет затем помещен в ТАБЛИЦУ ЗАДАЧ в поле 3 блока управления, которая формируется из шага задания.

Если вызов осуществлен с консоли, то запоминается ее номер; если вызов от других задач, то устанавливается номер главной консоли. После этого ПЛАНИРОВЩИК проверяет, впервые ли обрабатывается текущий вход из очереди заданий. Если вход обрабатывается впервые, то информация о нем помещается в таблице рангов. В ней хранятся ранги заданий, номера входов в очередь заданий и другая информация, на основании которой будет определен приоритет задачи, порожденной данным заданием. Так как ПЛАНИРОВЩИК является программой *P2*, то выполнение может быть прервано. Программа ПЛАНИРОВЩИК может сама прервать собственное выполнение для выдачи сообщения оператору, например об установке и включении требуемых устройств. Поэтому некоторые входы из очереди заданий уже ею анализировались и информация об их ранге уже помещена в соответствующую таблицу.

После этого ПЛАНИРОВЩИК проверяет, есть ли еще входы в очереди заданий данного потока. Если очередь исчерпана, то подготавливается к обработке очередь следующего потока, и ПЛАНИРОВЩИК переходит к анализу таблицы рангов. Просматривая таблицу рангов, ПЛАНИРОВЩИК выбирает задания с рангом, равным девяти. Эти задания должны быть выполнены в первую очередь, им отводятся все наличные ресурсы, и если их не хватает, эти ресурсы не перераспределяются. По мере освобождения новых ресурсов они выделяются заданию с рангом больше девяти. Так продолжается до тех пор, пока заданию с рангом больше девяти не будут выделены все требуемые ему ресурсы.

Если заданий с рангом девять нет, то выбирается задание с наивысшим рангом. Для выбранного задания проверяется, достаточно ли ему имеющихся в наличии устройств. Если устройств недоста-

точно, то подготавливается выбор следующего задания с помощью таблицы рангов. Если устройств достаточно, ПЛАНИРОВЩИК проверяет по ТАБЛИЦЕ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ, установлены эти устройства или нет. Если все или часть устройств не установлены, оператору выдается сообщение о включении и установке требуемых устройств. Если требуемые устройства включены, то проверяется по СПИСКУ РАСПРЕДЕЛЕННОЙ ПАМЯТИ, достаточно ли выбранному заданию имеющейся свободной основной памяти. Если наличного объема памяти недостаточно, управление передается на выбор и обработку следующего задания. Если требуемая память в наличии, то производится повторная проверка наличия включенных устройств, требуемых этому заданию, наличия основной памяти, дополнительная проверка, не появилось ли задание с рангом больше девяти. Если все в порядке, то производится окончательная модификация ТАБЛИЦЫ ИНФОРМАЦИИ об УСТРОЙСТВАХ, ТАБЛИЦЫ ПОТОКА, уточняются таблицы высокой памяти и их адреса в проблемной программе, которую требует выбранное задание, и ПЛАНИРОВЩИК вызывает ГЛАВНЫЙ ПЛАНИРОВЩИК. Если обнаружены какие-либо несоответствия при повторной проверке, управление передается на выбор и обработку следующего задания.

Во время своего выполнения по мере необходимости ПЛАНИРОВЩИК формирует сообщения в СИСТЕМНЫЙ ЖУРНАЛ.

15.4. Главный планировщик

ГЛАВНЫЙ ПЛАНИРОВЩИК — привилегированная программа *P*1 УПРАВЛЕНИЯ ЗАДАНИЯМИ. Она вызывается ПЛАНИРОВЩИКОМ на свое место, т. е. является транзитом ПЛАНИРОВЩИКА. Других способов вызова ГЛАВНОГО ПЛАНИРОВЩИКА не существует.

Исходными данными для программы ГЛАВНЫЙ ПЛАНИРОВЩИК являются таблица рангов, таблица параметров файлов, входящая в состав таблиц высокой памяти проблемной программы, таблица потоков. Результаты своей работы ГЛАВНЫЙ ПЛАНИРОВЩИК записывает в СИСТЕМНЫЙ ЖУРНАЛ и является единственной программой, которая выводит СИСТЕМНЫЙ ЖУРНАЛ на медленные устройства.

ГЛАВНЫЙ ПЛАНИРОВЩИК выполняет следующие функции:

упорядочивает и модифицирует таблицу рангов;

распределяет медленные и специфические устройства между задачами, модифицирует таблицу параметров файлов, вводя туда действительную мнемонику устройств;

проверяет, есть ли в наличии устройства с прямым доступом, и если надо, требует от оператора установить нужный логический том;

формирует ОБЛАСТЬ СВЯЗИ СУПЕРВИЗОРа для формируе-

мой задачи и размещает ключ защиты памяти в ТАБЛИЦЕ ЗАДАЧ в поле И;

уменьшает число ресурсов потока в соответствии с требованиями задач, выполняемых в этом потоке;

вызывает ЗАГРУЗЧИК для загрузки программы, подчиненной сформированной задаче.

Чтобы выбрать задание из очереди на выполнение, ПЛАНИРОВЩИКом создается таблица рангов. В таблице рангов размещаются задания, для которых есть в наличии все ресурсы. На основании таблицы рангов ГЛАВНЫЙ ПЛАНИРОВЩИК выбирает задание с наивысшим рангом для того, чтобы сформировать задачу для выполнения программы, указанной в задании. Как только задача начала существовать, она начинает «бороться» за ресурсы системы. Ей необходимо отвести медленные и быстродействующие устройства, область основной памяти для вызова подчиненной программы, загрузить подчиненную программу в основную память. Задача считается активизированной, как только управление передано подчиненной программе.

Функциональная блок-схема программы ГЛАВНЫЙ ПЛАНИРОВЩИК приведена на рис. 15.5.

Главный планировщик выбирает на основании таблицы рангов задание из очереди заданий и выделяет ему в ТАБЛИЦЕ ЗАДАЧ свободный участок, начиная тем самым инициирование задачи. Затем он проверяет, сформирована ли предыдущими программами УПРАВЛЕНИЯ ЗАДАНИЯМИ таблица параметров файлов, которая является одной из таблиц высокой памяти (см. рис. 15.2). Если такой таблицы еще нет, то она формируется, и ГЛАВНЫЙ ПЛАНИРОВЩИК проверяет, установлены ли устройства с прямым доступом, соответствующие логическим томам, используемым задачей. Если они еще не установлены или не включены, оператору выдается сообщение с требованием установить нужные тома. Затем задаче выделяются требуемые медленные устройства из числа устройств, зарезервированных ПЛАНИРОВЩИКом.

Выделение медленных устройств означает очередную модификацию таблицы информации об устройствах и таблицы потоков. Теперь туда помещается номер точки входа задачи, которой эти устройства выделены.

После этого производится анализ состояния подчиненной программы. Определяется, находится ли она в одной из библиотек системного файла или во входном потоке заданий. Если она находится во входном потоке заданий, то вызывается ПЛАНИРОВЩИК как транзит ГЛАВНОГО ПЛАНИРОВЩИКА для размещения программы в одной из библиотек системы. Если программа размещена в одной из библиотек системы, то проверяется, запуск программы производится с контрольной или начальной точки. Если запуск производится с начальной точки, то выполняются действия, связанные с подготовкой загрузки программ в основную память. Формируется как бы скелет области основной памяти, отведенной задаче. Произ-

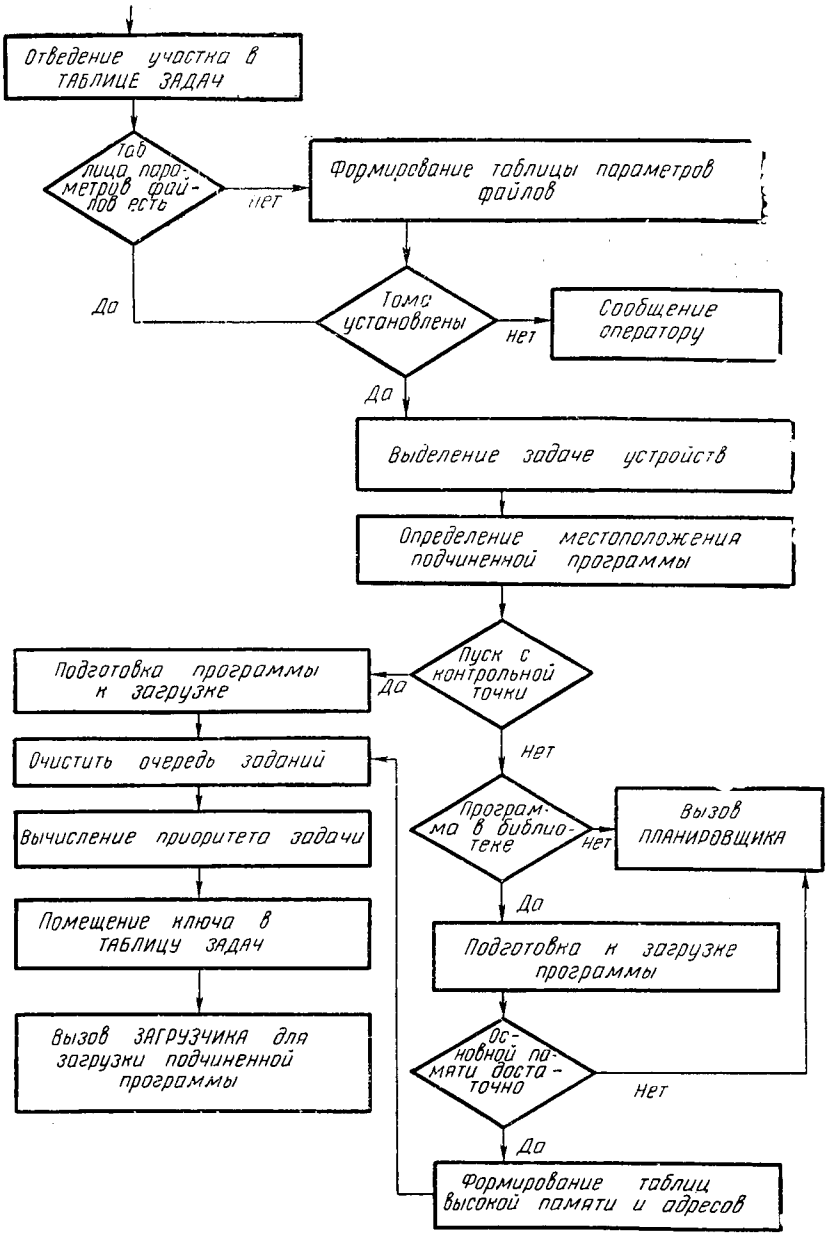


Рис. 15.5. ГЛАВНЫЙ ПЛАНИРОВЩИК

водятся различные уточнения существующих таблиц и вновь проверяется, достаточен ли объем основной памяти, выделенный ПЛАНИРОВЩИКОМ, для загрузки программы. Если памяти недостаточно, то управление передается ПЛАНИРОВЩИКу, который вызывается как транзит ГЛАВНОГО ПЛАНИРОВЩИКа.

Если условия выполнены, то производится точное вычисление действительных адресов и формирование таблиц высокой памяти, в которых мнемонические названия устройств заменяются их адресами. Задание, которое было активизировано для выполнения, вычеркивается из таблицы рангов и из очереди заданий. На основании ранга задания и существующей таблицы приоритетов проблемных задач определяется истинный приоритет задачи, который размещается в таблице приоритетов проблемных задач.

Выделенный ПЛАНИРОВЩИКОМ ключ защиты памяти размещается в поле И блока управления задачей в ТАБЛИЦЕ ЗАДАЧ. ГЛАВНЫЙ ПЛАНИРОВЩИК подготавливает вызов ЗАГРУЗЧИКа для загрузки подчиненной программы и передачи ей управления. Делается это обычным способом по ТАБЛИЦЕ ЗАДАЧ. ГЛАВНЫЙ ПЛАНИРОВЩИК задает соответствующие номера вызываемой и вызывающей задач для обращения к СУПЕРВИЗОРу и вызова программы СОГЛАСОВАНИЕ, которая сформирует в ТАБЛИЦЕ ЗАДАЧ соответствующую цепочку связей и передаст управление программе ВЫБОР. Программа ВЫБОР найдет требуемую системную задачу (в нашем случае ЗАГРУЗЧИК) и передаст ей управление. Загрузив программу, подчиненную нашей задаче, которая до этого момента была помечена как ожидающая, ЗАГРУЗЧИК передаст управление программе ВОЗВРАТ. Эта программа очистит цепочку связей в ТАБЛИЦЕ ЗАДАЧ, пометит задачу как не ожидающую и передаст управление программе ВЫБОР. Программа ВЫБОР определит задачу с наивысшим приоритетом и передаст ей управление. Такой задачей может быть и только что сформированная.

15.5. Перераспределитель

Программа ПЕРЕРАСПРЕДЕЛИТЕЛЬ выполняет функции управления заданиями, но является программой *Р2* СУПЕРВИЗОРа и не принадлежит УПРАВЛЕНИЮ ЗАДАНИЯМИ. Она состоит из двух фаз. Вторую фазу ПЕРЕРАСПРЕДЕЛИТЕЛЯ мы уже рассмотрели. Она предназначена для вызова ПЛАНИРОВЩИКА. Основными функциями первой фазы ПЕРЕРАСПРЕДЕЛИТЕЛЯ являются:

перераспределение освободившихся медленных устройств и модификация ТАБЛИЦЫ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ;

перераспределение основной памяти, освободившейся в результате работы как системных программ, так и обрабатывающих, и модификация СПИСКА РАСПРЕДЕЛЕННОЙ ПАМЯТИ;

перераспределение освободившихся ключей защиты памяти и модификация СПИСКА РАСПРЕДЕЛЕННОЙ ПАМЯТИ;

освобождение участков в ТАБЛИЦЕ ЗАДАЧ для тех задач, которые окончили свое выполнение, и модификация ТАБЛИЦЫ ЗАДАЧ, таблицы приоритетов проблемных задач;

попытка вызвать ПЛАНИРОВЩИК для инициирования новых заданий.

Существуют два способа вызова первой фазы ПЕРЕРАСПРЕДЕЛИТЕЛЯ: от макрокоманды CLOSE (ЗАКРЫТЬ), которая освобождает медленные устройства, и от программы СУПЕРВИЗОРа, обрабатывающей конец задания. В обоих случаях вызов осуществляется посредством команды обращения к СУПЕРВИЗОРу.

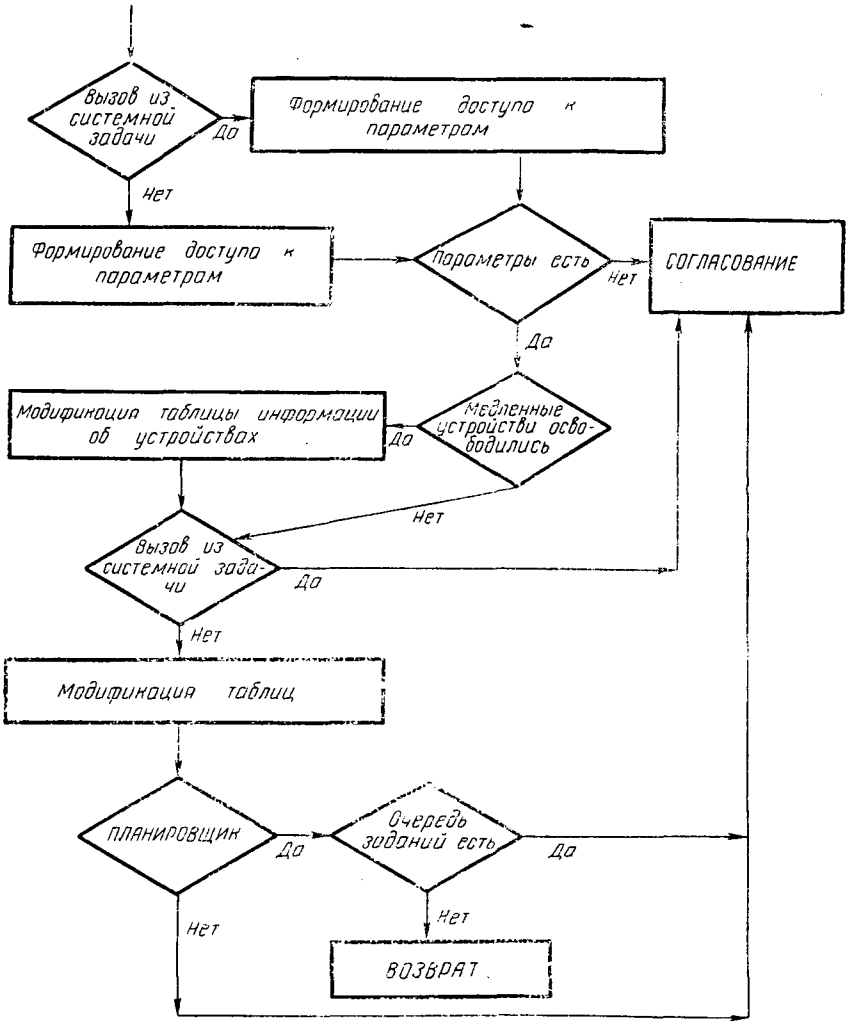


Рис. 15.6. Первая фаза ПЕРЕРАСПРЕДЕЛИТЕЛЯ

Выход из ПЕРЕРАСПРЕДЕЛИТЕЛЯ осуществляется либо обычным путем посредством передачи управления программе ВОЗВРАТ, либо посредством команды обращения к СУПЕРВИЗОРу для вызова СОГЛАСОВАНИЯ, указав в качестве вызываемой задачи номер точки входа в ТАБЛИЦУ ЗАДАЧ второй фазы ПЕРЕРАСПРЕДЕЛИТЕЛЯ для активизации ПЛАНИРОВЩИКА.

Функциональная блок-схема первой фазы ПЕРЕРАСПРЕДЕЛИТЕЛЯ приведена на рис. 15.6.

Так как параметры, необходимые для работы первой фазы ПЕРЕРАСПРЕДЕЛИТЕЛЯ, задаются по-разному в зависимости от способа обращения, то в первую очередь анализируется, откуда последовал вызов первой фазы, прежде чем будет сформулирован доступ к параметрам. В качестве параметров будут перечислены освободившиеся медленные устройства и номера точек входов задач, которые закончили свое выполнение. Параметров может и не быть. В этом случае делается попытка вызвать вторую фазу ПЕРЕРАСПРЕДЕЛИТЕЛЯ для инициирования ПЛАНИРОВЩИКА. Вызов осуществляется обычным путем посредством команды обращения к СУПЕРВИЗОРу, среди параметров которой в качестве вызываемой задачи указан номер точки входа второй фазы ПЕРЕРАСПРЕДЕЛИТЕЛЯ.

Если среди параметров имеется список освобожденных медленных устройств, то производится модификация ТАБЛИЦЫ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ. Вновь анализируется, откуда последовал вызов первой фазы ПЕРЕРАСПРЕДЕЛИТЕЛЯ. Если вызов последовал из системных задач, то предпринимается попытка вызвать вторую фазу ПЕРЕРАСПРЕДЕЛИТЕЛЯ для активизации ПЛАНИРОВЩИКА. Если вызов был из проблемной задачи, то производится модификация ТАБЛИЦЫ ЗАДАЧ для того, чтобы освободить участок, занимаемый блоком управления задачей, окончившей выполнение. Одновременно модифицируется СПИСОК РАСПРЕДЕЛЕННОЙ ПАМЯТИ и освобождается ключ защиты памяти, а также аннулируется область связи СУПЕРВИЗОРа для данной задачи.

В качестве задачи, которая закончила свое выполнение, может выступать ПЛАНИРОВЩИК. Для него делается проверка, есть ли еще очереди заданий. Если таковые есть, то вновь предпринимается попытка вызвать ПЛАНИРОВЩИК обычным путем. Если задача, закончившая выполнение, не является ПЛАНИРОВЩИКОМ, то первая фаза ПЕРЕРАСПРЕДЕЛИТЕЛЯ закончила свою работу и, следовательно, обязана передать управление программе ВОЗВРАТ.

15.6. Завершение выполнения задания

Программа СУПЕРВИЗОРа *P2* ОБРАБОТКА КОНЦА ЗАДАНИЯ получает управление при нормальном и анормальном завершении выполнения задачи. Появление причины анормального за-

вершения задачи означает, что во время выполнения подчиненной программы возникли условия, делающие невозможным ее продолжение. Причинами анормального завершения задачи могут быть программные сбои, неисправимые сбои ввода-вывода, требование оператора о прекращении выполнения задания, сообщения самой подчиненной программы о невозможности ее выполнения.

Анормальное завершение задачи воспринимается как окончание действий, указанных в шаге задания, из которого была инициирована задача. Поэтому ОБРАБОТКА КОНЦА ЗАДАНИЯ вызывает ПЛАНИРОВЩИК обычным путем для инициирования выполнения следующего шага текущего или нового задания.

Анормальное завершение задачи вызывает прекращение выполнения всего задания независимо от того, сколько шагов осталось невыполненным. Задание снимается с обработки, и в системный журнал выводится сообщение о причине прекращения всего задания. Оставшиеся шаги задания не выполняются. Оператору выдается сообщение о прекращении выполнения задания. При этом оператор может потребовать распечатать содержимое областей основной памяти, используемых управляющей и проблемной программами, для анализа причины прекращения выполнения задания. Такая распечатка называется разгрузкой памяти.

Как указывалось, прекращение выполнения задачи может быть вызвано требованием, поступившим из самой проблемной программы, когда она обнаружила, что дальнейшее выполнение не имеет смысла. Это требование выдается в виде специальных макрокоманд, в которых указано, надо или не надо производить разгрузку памяти.

После выполнения действий, предпринятых по анормальному завершению задачи, в область проблемной задачи вызывается ПЛАНИРОВЩИК обычным путем для инициирования следующего задания.

ВОПРОСЫ К ГЛАВЕ

1. Расскажите порядок прохождения задания через вычислительную систему.
2. Какие управляющие таблицы используются программами УПРАВЛЕНИЕ ЗАДАНИЯМИ?
3. Относится ли ПЕРЕРАСПРЕДЕЛИТЕЛЬ к собственным программам УПРАВЛЕНИЯ ЗАДАНИЯМИ?
4. Какие программы СУПЕРВИЗОРа и УПРАВЛЕНИЯ ЗАДАНИЯМИ будут работать, чтобы вызвать ПЛАНИРОВЩИК?
5. Какая программа формирует задачу из шага задания?
6. Назовите причины анормального завершения задачи.
7. Какая программа формирует очередь заданий?
8. С какими библиотеками оперируют программы УПРАВЛЕНИЯ ЗАДАНИЯМИ?
9. Объясните связь между заданиями, программой и задачей?
10. Какие задачи (проблемные или системные) формируются для программ УПРАВЛЕНИЯ ЗАДАНИЯМИ?

Глава 16

СИСТЕМА ПОДГОТОВКИ ПРОГРАММ

16.1. Прохождение программ через вычислительную систему

Группа программ, объединенных под общим названием СИСТЕМА ПОДГОТОВКИ ПРОГРАММ, представляет программисту ряд удобств при разработке и составлении программ. Основное назначение СИСТЕМЫ ПОДГОТОВКИ ПРОГРАММ — обеспечить сборку и прохождение через вычислительную систему программ на исходном языке, включая трансляцию, редактирование и отладку. Она обеспечивает этапность подготовки программы к выполнению и представляет программисту возможность: разбить программу на части, для каждой из частей выбрать наиболее подходящий язык из имеющихся в системе алгоритмических языков или языков программирования; автономно протранслировать каждую часть; полученные объектные модули каждой части отредактировать, объединяя их в одну готовую к выполнению программу.

Связи, существующие между различными автономными частями программы, определяются программистом символически на исходных языках: как внешние ссылки они сохраняются в объектных модулях и получают свои конкретные значения только на этапе редактирования.

При создании СИСТЕМЫ ПОДГОТОВКИ ПРОГРАММ большое внимание было уделено модульности программ и возможности связывать отдельные части программ различными способами. Были усовершенствованы используемые ранее методы сборки программ. Для всех программ независимо от исходного языка, на котором они написаны, стандартизовано размещение их в библиотеках системы на любом этапе подготовки программ к выполнению. Все программы могут быть загружены и выполнены в любом месте основной памяти. Их местонахождение в основной памяти определяется в момент загрузки.

Отдельно написанные куски большой программы могут быть объединены в одну программу на любом из четырех этапов прохождения программы в вычислительной системе: на этапе компиляции, на этапе редактирования, во время ввода задания, во время выполнения задачи.

Во время работы компилятора на его входе можно объединить колоды перфокарт различных подпрограмм, написанных на исходном языке. На этапе редактирования загрузочные модули и отдельно скомпилированные объектные модули могут быть переработаны РЕДАКТОРОМ в один загрузочный модуль. При этом модуль не обязательно должен целиком размещаться в основной памяти во время его выполнения.

На этапе ввода задания отдельным шагам задания могут соот-

ветствовать части большой программы, которые оформляются программой ГЛАВНЫЙ ПЛАНИРОВЩИК в самостоятельные задачи. Связь между такими частями программы осуществляет ПЛАНИРОВЩИК. Подобная техника объединения частей программ в единую приводит к значительной экономии как основной памяти, так и вспомогательной на устройствах с прямым доступом. Объясняется это тем, что в каждый момент времени необходима память только для одного шага задания, когда он инициируется как задача.

На этапе выполнения задачи один загрузочный модуль, указанный в шаге задания, может обратиться к другим загрузочным модулям. Вызываемые модули размещаются в свободной области основной памяти и динамически связываются с вызывающим модулем. При этом большие загрузочные модули могут во время выполнения перекрываться, а отдельные из них иногда могут совместно использоваться различными задачами в мультипрограммном режиме работы.

В различных операционных системах СИСТЕМА ПОДГОТОВКИ ПРОГРАММ относится либо к УПРАВЛЕНИЮ ЗАДАНИЯМИ, либо к системным обслуживающим программам, либо выделяется как самостоятельный раздел операционной системы. Мы будем рассматривать СИСТЕМУ ПОДГОТОВКИ ПРОГРАММ как самостоятельный компонент операционной системы.

Основной единицей, с которой оперирует СИСТЕМА ПОДГОТОВКИ ПРОГРАММ, является программный модуль. Имеются три типа программных модулей: исходный, объектный и загрузочный. Часть программы или программа, представляющая собой набор операторов исходного языка и предназначенная для ввода и переработки компилятором с этого исходного языка, называется *исходным модулем*. Исходные модули хранятся в библиотеке исходных модулей.

Часть программы или программа, полученная в результате работы компилятора по переработке одного или нескольких исходных модулей, называется *объектным модулем*. Объектные модули размещаются в библиотеке объектных модулей. Они еще не готовы к выполнению.

Объектные модули, прошедшие этап редактирования, образуют *загрузочные модули,* которые готовы к выполнению и всегда размещаются в библиотеке загрузочных модулей. Загрузочный модуль может формироваться из любого числа объектных модулей и ранее сформированных загрузочных модулей. Это позволяет ранее полученную программу дополнять новыми подпрограммами, формировать программы более сложной структуры, вносить изменения в ранее сформированные загрузочные модули без повторной компиляции всей программы, отдельно отлаживать различные подпрограммы одной большой программы.

Каждый модуль может состоять из нескольких сегментов. Сегмент представляет собой последовательность команд, которая не зависит от расположения других аналогичных последовательностей.

Сегменты являются единицами, которые могут исключаться, заменяться или добавляться в один модуль.

Этапы подготовки программы к выполнению иллюстрируются рис. 16.1.

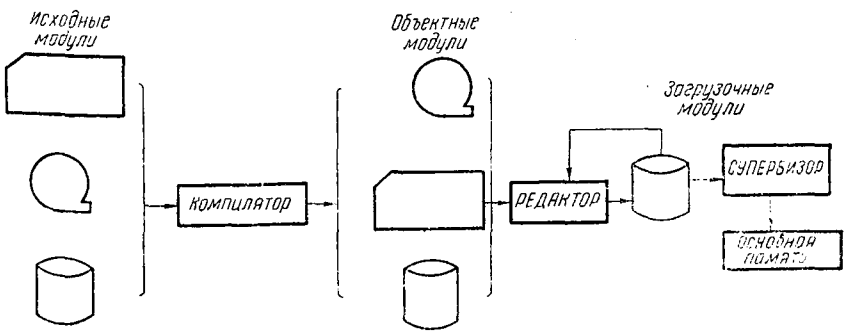


Рис. 16.1. Подготовка программ

Исходные модули, которые могут находиться на различных носителях (на перфокартах, магнитных лентах в библиотеке исходных модулей на устройстве с прямым доступом), обрабатываются компиляторами с исходного языка, образуя объектные модули, которые также могут быть выведены компилятором на перфокарты, магнитную ленту или размещены в библиотеке объектных модулей на устройстве с прямым доступом. Объектные модули обрабатываются программой РЕДАКТОР, одной из основных программ СИСТЕМЫ ПОДГОТОВКИ ПРОГРАММ и образуют загрузочные модули, которые размещаются в библиотеке загрузочных модулей на устройстве с прямым доступом. Загрузочные модули вызываются СУПЕРВИЗОРом для выполнения в основную память при инициировании задачи, которой подчинена программа.

16.2. Структура программ

Проблемная программа состоит из одного или нескольких загрузочных модулей. При создании задачи, которая инициируется из одного шага задания, в управляющем операторе EXEC, определяющем начало шага задания, указывается имя первого загрузочного модуля, который должен быть выполнен. Загрузочный модуль с этим именем может быть введен в основную память целиком или частями, может быть связан с другими загрузочными модулями во время выполнения в зависимости от структуры программы, которая задается программистом на исходном языке. Программа может иметь простую структуру, с запланированным перекрытием, динамическую последовательную структуру или динамическую параллельную структуру.

Программа, имеющая простую структуру, состоит из единственного загрузочного модуля, который содержит все необходимое для выполнения задачи и загружается в основную память как единое целое. При этом не имеет значения, был ли этот модуль составлен РЕДАКТОРОМ из нескольких загрузочных модулей и будет ли задача во время выполнения обращаться к СУПЕРВИЗОРу с требованием вызвать другие загрузочные модули.

Программа, имеющая структуру с запланированным перекрытием, состоит также из одного загрузочного модуля. Но в отличие от простой структуры загрузочный модуль с запланированным перекрытием поделен на сегменты, которые могут загружаться в основную память неодновременно. Эти сегменты являются транзитами по отношению друг к другу и вызываются на одно и то же место основной памяти. Программы с запланированным перекрытием более экономно используют основную память, чем программы с простой структурой. Структура с запланированным перекрытием иллюстрируется на рис. 16.2.

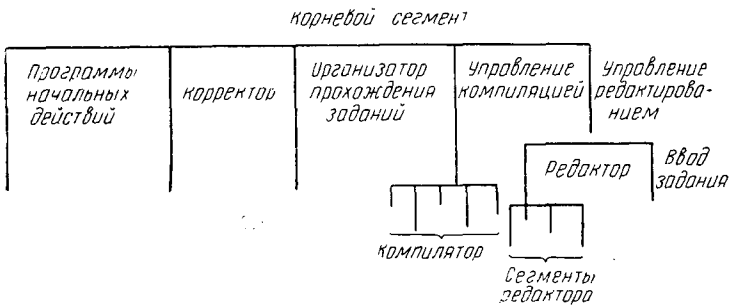


Рис. 16.2. Структура системы подготовки программ

Программа, имеющая динамическую последовательную структуру, состоит из нескольких загрузочных модулей. Каждый модуль может иметь простую структуру или с запланированным перекрытием. Загрузочные модули загружаются последовательно на одно и то же место основной памяти, т. е. являются транзитами по отношению друг к другу. Все связи между отдельными загрузочными модулями устанавливаются СУПЕРВИЗОРом, который в данном случае играет роль корневого сегмента.

Программа, имеющая динамическую параллельную структуру, также состоит из нескольких загрузочных модулей, имеющих любую структуру. Программы с динамической параллельной структурой могут создаваться только в системах мультипрограммирования с переменным числом задач. Как и для программ с динамической последовательной структурой, СУПЕРВИЗОР играет роль корневого сегмента для вызова отдельных загрузочных модулей в основную память для выполнения. В этом случае допускается параллель-

ное выполнение двух и более модулей, т. е. загрузочные модули программы с динамической параллельной структурой не обязательно будут вызываться на одно и то же место основной памяти и являться транзитами.

16.3. Состав и структура системы подготовки программ

СИСТЕМА ПОДГОТОВКИ ПРОГРАММ состоит из ПРОГРАММЫ НАЧАЛЬНЫХ ДЕЙСТВИЙ, ОРГАНИЗАТОРА ПРОХОЖДЕНИЯ ЗАДАНИЙ, КОРРЕКТОРа, РЕДАКТОРа, УПРАВЛЕНИЯ КОМПИЛЯЦИЕЙ И РЕДАКТИРОВАНИЕМ. Структура СИСТЕМЫ ПОДГОТОВКИ ПРОГРАММ приведена на рис. 16.2.

Все программы СИСТЕМЫ ПОДГОТОВКИ являются защищенными программами *P*1. СИСТЕМА ПОДГОТОВКИ ПРОГРАММ вызывается обычным для программ *P*1 путем по ТАБЛИЦЕ ЗАДАЧ, когда она инициируется УПРАВЛЕНИЕМ ЗАДАНИЯМИ как задача. Приоритет отдельных программ СИСТЕМЫ ПОДГОТОВКИ ПРОГРАММ может динамически изменяться во время выполнения. УПРАВЛЕНИЕ ЗАДАНИЯМИ инициирует систему подготовки как задачу, когда обрабатываемый ею шаг задания содержит управляющие параметры языка управления заданиями, предназначенные для вызова РЕДАКТОРа, КОРРЕКТОРа или КОМПИЛЯТОРА с одного из исходных языков.

При инициировании СИСТЕМЫ ПОДГОТОВКИ ПРОГРАММ как задачи управление передается корневому сегменту. Вызов и управление всеми другими программами осуществляет сама СИСТЕМА ПОДГОТОВКИ. Это значит, что в ТАБЛИЦЕ ЗАДАЧ для СИСТЕМЫ ПОДГОТОВКИ ПРОГРАММ отводится всего только один блок управления задачей для корневого сегмента.

Корневой сегмент представляет собой программу, которая всегда находится в основной памяти во время выполнения СИСТЕМЫ ПОДГОТОВКИ ПРОГРАММ. Он содержит ОБЛАСТЬ ПАМЯТИ СУПЕРВИЗОРа, таблицы определения файлов и команды вызова программы НАЧАЛЬНЫХ ДЕЙСТВИЙ. Все остальные программы являются транзитными и, окончив свои действия, вызывают на свое место в основной памяти ОРГАНИЗАТОР ПРОХОЖДЕНИЯ ЗАДАНИЙ.

Последовательность вызова отдельных программ СИСТЕМЫ ПОДГОТОВКИ ПРОГРАММ представлена на рис. 16.3.

При обращении к СИСТЕМЕ ПОДГОТОВКИ ПРОГРАММ управление передается корневому сегменту, который вызывает программу начальных действий. Назначение этой программы — подготовить необходимую информацию для работы СИСТЕМЫ ПОДГОТОВКИ ПРОГРАММ. Она определяет входные и выходные файлы системы, открывает их, модифицирует соответственно таблицы определения файлов, хранящиеся в корневом сегменте, определяет, с какими библиотеками системы будет работать РЕДАКТОР, где

находится входной поток заданий, который будет обрабатываться программой ОРГАНИЗАТОР ПРОХОЖДЕНИЯ ЗАДАНИЙ. Вся информация, которую подготавливает программа начальных действий, размещается в корневом сегменте. По окончании начальных действий программа вызывает на свое место ОРГАНИЗАТОР ПРОХОЖДЕНИЯ ЗАДАНИЙ и передает ему управление.

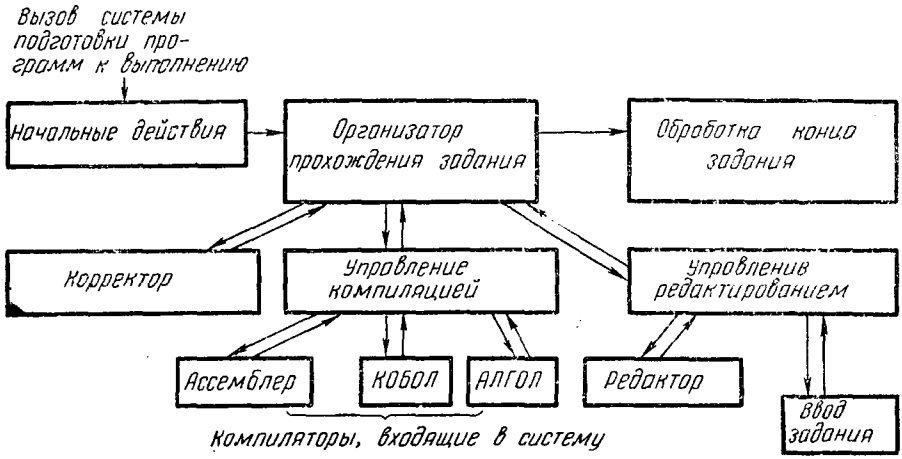


Рис. 16.3. Последовательность выполнения программ системы подготовки

Назначение ОРГАНИЗАТОРА ПРОХОЖДЕНИЯ ЗАДАНИЙ — определить на основании операторов, описывающих задание, программу, которой следует передать управление, проверить, можно ли передать этой программе управление, закрыть файлы, которые были открыты программой начальных действий. Если нельзя передать управление ни одной из программ СИСТЕМЫ ПОДГОТОВКИ в силу обнаружения ошибок или если пришла управляющая карта, определяющая конец задания, ОРГАНИЗАТОР ПРОХОЖДЕНИЯ ЗАДАНИЙ передает управление программе *Р2* СУПЕРВИЗОРа ОБРАБОТКА КОНЦА ЗАДАНИЙ.

При появлении управляющей карты, требующей внести изменения в исходный модуль какой-либо программы, находящейся в библиотеке исходных модулей, ОРГАНИЗАТОР ПРОХОЖДЕНИЯ ЗАДАНИЙ вызывает программу КОРРЕКТОР на свое место в основной памяти. Назначение КОРРЕКТОРа — поддержание в порядке исходных модулей библиотеки исходных модулей. С помощью КОРРЕКТОРа можно вставить, вычеркнуть, заменить целый исходный модуль в библиотеке исходных модулей. Можно произвести изменения внутри модуля: вставить, заменить, вычеркнуть или переместить одну или несколько карт программы на исходном языке. КОРРЕКТОР также позволяет вычеркнуть один или несколько объектных модулей из библиотеки объектных модулей. По оконча-

нии своих действий КОРРЕКТОР вызывает на свое место в основной памяти ОРГАНИЗАТОР ПРОХОЖДЕНИЯ ЗАДАНИЯ.

При появлении управляющей карты, требующей скомпилировать программу с какого-либо исходного языка, ОРГАНИЗАТОР ПРОХОЖДЕНИЯ ЗАДАНИЙ вызывает на свое место программу УПРАВЛЕНИЕ КОМПИЛЯЦИЕЙ. Назначение этой программы — проверить правильность вызова компилятора, убедиться в том, что для него имеется достаточно основной памяти, найти КОМПИЛЯТОР в библиотеке системы, вызвать корневой сегмент компилятора в основную память и передать ему управление. После окончания работы КОМПИЛЯТОРА он обязан передать управление программе УПРАВЛЕНИЕ КОМПИЛЯЦИЕЙ.

УПРАВЛЕНИЕ КОМПИЛЯЦИЕЙ определяет время работы компилятора, условия компиляции, формирует сообщение в СИСТЕМНЫЙ ЖУРНАЛ. Если компиляция прошла успешно, исходный модуль, который обрабатывался компилятором, вычеркивается из библиотеки исходных модулей, а полученный объектный модуль размещается в библиотеке объектных модулей. При этом, если в библиотеке объектных модулей уже имелся объектный модуль с этим же именем, что и полученный, старый объектный модуль вычеркивается из библиотеки.

Затем программа УПРАВЛЕНИЕ КОМПИЛЯЦИЕЙ проверяет, какая следует управляющая карта. Если следующая управляющая карта определяет вызов компилятора, то УПРАВЛЕНИЕ КОМПИЛЯЦИЕЙ повторяет свои действия для вызова требуемого компилятора. При появлении любой другой карты или отсутствии таковых УПРАВЛЕНИЕ КОМПИЛЯЦИЕЙ вызывает на свое место ОРГАНИЗАТОР ПРОХОЖДЕНИЯ ЗАДАНИЯ.

При появлении управляющей карты, требующей редактирования программы, ОРГАНИЗАТОР ПРОХОЖДЕНИЯ ЗАДАНИЯ вызывает на свое место УПРАВЛЕНИЕ РЕДАКТИРОВАНИЕМ. УПРАВЛЕНИЕ РЕДАКТИРОВАНИЕМ является корневым сегментом РЕДАКТОРА.

РЕДАКТОР (иногда его называют РЕДАКТОРОМ СВЯЗЕЙ или ФОРМИРОВАТЕЛЕМ) является наиболее сложным элементом СИСТЕМЫ ПОДГОТОВКИ ПРОГРАММ. Его назначение — объединить отдельные сегменты и объектные модули в единый загрузочный модуль, сформировать связи внутри загрузочного модуля между составляющими его сегментами, установить связи с другими загрузочными модулями, входящими в одну программу. РЕДАКТОР также включает в загрузочный модуль контрольные точки, входы обращения к отладочным программам, входы обращения к стандартным функциям из библиотеки стандартных функций.

Контрольная точка предназначена для разрешения временного прекращения программы или минимизации потери времени из-за сбоя машины или какой-либо ошибки. Она позволяет осуществить пуск программы не с начала. Формирование контрольных точек происходит при появлении макрокоманды СНКРТ. При выполнении

этой макрокоманды запоминается управляющая информация задачи и те области основной памяти, которые необходимы для повторения программы с контрольной точки. Происходит как бы фотографирование результатов выполнения программы.

Фотографирование осуществляется СУПЕРВИЗОРом. РЕДАКТОР вставляет в проблемную программу на место макрокоманды СНКРТ команды обращения к СУПЕРВИЗОРу. После того как СУПЕРВИЗОР выполнит фотографирование нужной информации, управление возвращается проблемной программе, которая продолжает свои вычисления. Результаты фотографирования при необходимости можно отпечатать с помощью отладочных программ.

Если в результате какой-либо ошибки появилась необходимость повторить выполнение программы, то ее выполнение можно начать с контрольной точки, после которой возникла ошибка. При повторении ГЛАВНЫЙ ПЛАНИРОВЩИК найдет информацию, сфотографированную в контрольной точке. На основании этой информации будет обеспечена правильная установка томов и подвод нужного места магнитных лент. Программа, выполняющая свои действия с контрольной точки, продолжает свою работу, словно она не прерывалась. За правильное размещение контрольных точек внутри программы отвечает программист. Они должны быть установлены в тех местах программы, где наиболее просто осуществить повторение, так как СУПЕРВИЗОР не фотографирует файлы данных.

Аналогично формированию контрольных точек РЕДАКТОР формирует входы обращения к отладочным программам. Вызов отладочных программ и их выполнение будет происходить в момент выполнения проблемной программы, когда встретится подобный вход.

РЕДАКТОР вставляет в проблемную программу входы обращения к стандартным функциям. Сами функции из библиотеки стандартных функций будут размещены в программе во время ее загрузки в основную память.

РЕДАКТОР состоит из четырех фаз, кроме того, к нему относятся программы включения и редактирования файлов проблемной программы. Первая фаза РЕДАКТОРа выполняет чтение управляющих параметров для работы РЕДАКТОРа, отбор всех объектных модулей из библиотеки объектных модулей, которые нужно объединить, определяет входы в отладочные программы и обращения к стандартным функциям из библиотеки стандартных функций. Вторая фаза РЕДАКТОРа вычисляет адреса и формирует связи внутри одного загрузочного модуля для составных сегментов. Практически основное формирование загрузочного модуля осуществляется второй фазой. Третья фаза предназначена для вывода отредактированной программы, т. е. для размещения загрузочного модуля в библиотеку загрузочных модулей, формирования связей между различными загрузочными модулями одной программы. При этом если в библиотеке загрузочных модулей уже находится модуль с тем же именем, что и вновь сформированный, то старый модуль вычерки-

вается из библиотеки. Четвертая фаза РЕДАКТОРа выполняется только при появлении макрокоманд, связанных со специальным режимом редактирования.

Программа ВКЛЮЧЕНИЕ ДАННЫХ обеспечивает проблемной программе ввод используемых ею данных, оформление их в виде файлов, размещение в библиотеке исходных модулей. Это осуществляется с целью ускорения прохождения проблемной программы, когда ее входные файлы находятся на перфокартах или перфолентах.

Программа РЕДАКТИРОВАНИЕ ФАЙЛОВ обеспечивает псевдовывод как для проблемных программ, так и для программ СИСТЕМЫ ПОДГОТОВКИ. Если занято выходное устройство, на которое необходимо вывести выходные файлы, программа РЕДАКТИРОВАНИЕ ФАЙЛОВ записывает их в требуемой форме в СИСТЕМНЫЙ ЖУРНАЛ для последующего вывода после освобождения устройства.

Если после окончания редактирования программы необходимо ее выполнить, УПРАВЛЕНИЕ РЕДАКТИРОВАНИЕМ вызывает ВВОД ЗАДАНИЙ, который размещает задание на выполнение отредактированной программы в очереди заданий. После окончания работы программы ВВОД ЗАДАНИЙ управление возвращается программе УПРАВЛЕНИЕ РЕДАКТИРОВАНИЕМ. Если больше нет управляющих карт, требующих редактирования программ, УПРАВЛЕНИЕ РЕДАКТИРОВАНИЕМ вызывает на свое место ОРГАНИЗАТОР ПРОХОЖДЕНИЯ ЗАДАНИЙ.

СИСТЕМА ПОДГОТОВКИ ПРОГРАММ может быть инициирована при необходимости связать отдельные объектные модули в один загрузочный модуль, чтобы подготовить программу к выполнению. Такой режим работы СИСТЕМЫ ПОДГОТОВКИ называется режимом организации. Кроме этого, во время выполнения проблемных программ может возникнуть необходимость обращения к отдельным программам СИСТЕМЫ ПОДГОТОВКИ. Обращение к ним проблемная программа осуществляет посредством вызова программы ПЛАНИРОВЩИК. Такой режим вызова СИСТЕМЫ ПОДГОТОВКИ называется режимом выполнения.

В режиме организации все программы СИСТЕМЫ ПОДГОТОВКИ выполняются под управлением ОРГАНИЗАТОРА ПРОХОЖДЕНИЯ ЗАДАНИЯ, как было описано выше. ОРГАНИЗАТОР ПРОХОЖДЕНИЯ ЗАДАНИЯ определяет управляющие параметры, поступающие в его распоряжение, и вызывает соответствующие программы из основной главы с библиотекой системы подготовки. В режиме организации можно, например, вставить исходный модуль в библиотеку исходных модулей, откорректировать уже существующий исходный модуль, откомпилировать и объединить эти два исходных модуля в один объектный модуль, а из нескольких объектных модулей сформировать один загрузочный модуль.

Таким образом, с функциональной точки зрения режим организации можно рассматривать как фазу редактирования библиотек:

исходных, объекгных и загрузочных модулей. Если в загрузочный модуль вставлены входы отладочных программ, то он размещается во временной библиотеке загрузочных модулей. Одновременно для такой программы вызывается ВВОД ЗАДАНИЙ для размещения в очереди заданий входа на выполнение и отладку отредактированной программы.

В режиме выполнения отдельные программы СИСТЕМЫ ПОДГОТОВКИ вызываются без помощи ОРГАНИЗАТОРА ПРОХОЖДЕНИЯ ЗАДАНИЯ. Например, если во время выполнения проблемной программы ей потребовались данные, которые не находятся в одной из библиотек системы, то проблемная программа вызывает ПЛАНИРОВЩИК. ПЛАНИРОВЩИК, в свою очередь, вызывает программу ВКЛЮЧЕНИЕ ДАННЫХ, относящуюся к РЕДАКТОРУ, и передает ей управление. ВКЛЮЧЕНИЕ ДАННЫХ вводит исходные данные проблемной программы, размещает их в библиотеке исходных модулей и возвращает управление прерванной проблемной программе для продолжения вычислений.

Аналогично осуществляется вызов программы РЕДАКТИРОВАНИЕ ФАЙЛОВ для организации псевдовывода по требованию любой программы, выполняющейся в системе. Таким образом, в режиме выполнения отдельные программы СИСТЕМЫ ПОДГОТОВКИ могут оформляться в виде самостоятельных задач.

ВОПРОСЫ К ГЛАВЕ

1. Можно ли программировать отдельные части программы на разных языках?
2. Как осуществляется обращение к библиотеке стандартных функций?
3. Сколько задач формируется для вызова программ СИСТЕМЫ ПОДГОТОВКИ в различных режимах работы?
4. Как получается загрузочный модуль?
5. Какую структуру может иметь программа?
6. Какая задача (системная или проблемная) формируется для вызова программ СИСТЕМЫ ПОДГОТОВКИ?

Глава 17

ЛОГИЧЕСКАЯ СИСТЕМА УПРАВЛЕНИЯ ВВОДОМ-ВЫВОДОМ

17.1. Обработка файлов

Совокупность способа организации файла и языка его обработки образует определенный метод доступа к файлу. К файлам, имеющим последовательную организацию, применим последовательный метод доступа. Таким образом, последовательный метод доступа может быть использован для обращения к файлам на медленных устройствах, магнитных лентах и некоторым файлам на устройствах с прямым доступом.

Для обращения к файлам данных, имеющих последовательную организацию, можно использовать базисный язык доступа и язык доступа с очередями. В результате получаем два метода последовательного доступа: базисный последовательный метод доступа и последовательный метод доступа с очередями.

Для каждого метода доступа существуют специальные макрокоманды с соответствующим набором параметров. Существуют и некоторые ограничения, запрещающие использование определенных, зависящих от типа устройства, макрокоманд. Например, макрокоманда CNTRL управляет работой медленных устройств ввода-вывода. С ее помощью можно, например, выбрать приемный карман на устройстве чтения перфокарт, указать число пропущенных строк при печати, управлять кареткой при выводе на перфоратор. Очевидно, что использовать эту макрокоманду для управления последовательным файлом на устройствах с прямым доступом нельзя.

Последовательный метод доступа дает возможность одинаковым образом обратиться к любому файлу, имеющему последовательную организацию, независимо от типа устройства, на котором расположен файл. Это свойство позволяет строить проблемные программы, независимые от типа устройств, которые будут в ней использоваться для выполнения операций ввода-вывода. Независимость обеспечивается и возможностями, которые предоставляются программисту языком управления заданиями. Все действительные характеристики файла, такие, как тип устройства, размер блока и записи, формат записи и другая информация, определяется с помощью оператора DD описания данных языка управления заданиями.

Все последовательные файлы, за исключением файлов на перфолентах, могут иметь любой формат записей (фиксированной, переменной или неопределенной длины). Файлы на перфолентах могут состоять из записей фиксированной или неопределенной длины.

При использовании базисного метода доступа на программисте лежит ответственность за правильную организацию буферизации. При использовании последовательного метода доступа с очередями буферизацию производит логическая система управления вводом-выводом. Файлы с последовательной организацией всегда имеют блочную структуру.

На устройствах с прямым доступом файлы могут иметь прямую организацию. Прямая организация файла предполагает наличие взаимосвязи между идентификацией каждой записи и ее местоположением в томе с прямым доступом. Эта взаимосвязь устанавливается программистом. При удачном выборе способа идентификации записи прямой метод доступа позволяет быстро обращаться к отдельной записи. Этот способ организации файлов наиболее эффективен в случаях, когда требуется обработка отдельных записей файла.

Файлы с прямой организацией можно обрабатывать только с помощью базисного языка доступа. По этой причине обмен между

основной памятью и устройством с прямым доступом осуществляется по одной записи. Если обмен необходимо производить блоками, то все операции, связанные с объединением записей в блоки и выделением записей из блоков, производятся программистом.

При разработке файлов с прямой организацией можно использовать прямую или косвенную адресацию записей. Прямая адресация предполагает знание местонахождения записи в файле. В каждой записи отводится поле, называемое ключом. Ключ представляет собой, например, порядковый номер записи в файле. Записи располагаются в последовательности ключей, для записей с пропущенными ключами оставляются свободные места. Недостаток такого способа в нерациональном использовании пространства вспомогательной памяти.

Другой способ прямой адресации, когда ключи выносятся из записей в отдельную справочную таблицу. Справочная таблица должна содержать действительные адреса записей на томе с прямым доступом и указание, к какой записи этот адрес относится. Этот способ прямой организации обладает недостатками: эффективное использование справочной таблицы возможно только по отношению к небольшим файлам данных; для поиска и исправления справочной таблицы требуется время.

Более сложный способ прямой организации файла предполагает использование косвенной адресации. При косвенной адресации адрес каждой записи определяется в результате математической манипуляции с ключом. Такую манипуляцию называют преобразованием ключа. Существует большое число возможных преобразований ключа.

Так как файлы с прямой организацией располагаются на устройстве с прямым доступом, они могут одновременно обрабатываться в программах нескольких задач. Чтобы предотвратить конфликтные ситуации при одновременном обращении к одной и той же записи (одна задача хочет записать модифицированную запись, а другая задача в это же время пытается прочитать эту запись), можно с помощью специальной макрокоманды предусмотреть монопольное управление файлом. При этом в блоке управления файлом (табл. 13.5) устанавливается маркер монопольного управления файлом.

Развитием последовательной организации на устройствах с прямым доступом является индексно-последовательная организация, которая позволяет обрабатывать файл последовательно или обращаться к отдельным записям файла. При этом можно производить вставку или удаление отдельных записей без перезаписи всего файла. Для обработки индексно-последовательного файла можно пользоваться как базисным языком, так и языком доступа с очередями.

Язык доступа с очередями используется при создании файла, обработке и исправлении. Для чтения файла или вставки новых записей в файл применяется базисный язык доступа. Отличаются эти

два метода доступа параметрами, используемыми в макрокомандах обращения к файлу. При базисном методе доступа к файлу буферизацию должен производить программист. При индексно-последовательном доступе с очередями буферизацию производит логическая система управления вводом-выводом.

Каждая запись файла с индексно-последовательной организацией содержит поле ключа. Записи в файле упорядочены по ключу и сблокированы. Каждому блоку записей предшествует поле ключа, которое соответствует ключу последней записи в блоке. При организации индексно-последовательного файла на томе с прямым доступом организуются две области: основная область и область переполнения. Основная область данных формируется в момент создания файла, т. е. при первоначальной записи файла на томе.

В это же время логическая система управления вводом-выводом формирует область индекса дорожек, в которой производит учет записей, содержащихся на каждой дорожке. Каждый элемент в области индекса дорожек идентифицирует ключ записи, которая была последней записана на данную дорожку. Таким образом, индекс каждой дорожки содержит ключ последней записи. Такой индекс формируется для каждой дорожки цилиндра. Если файл занимает несколько цилиндров, то формируется индекс более высокого уровня — индекс цилиндров. Отдельный элемент индекса цилиндров указывает ключ записи, которая является последней на данном цилиндре. Для увеличения скорости поиска в индексе цилиндров формируется главный индекс для заданного числа цилиндров. Так получается дерево индексов, на основе которого осуществляется быстрый доступ к записям файла. Структура дерева индексов для файлов с индексно-последовательной организацией приведена на рис. 17.1.

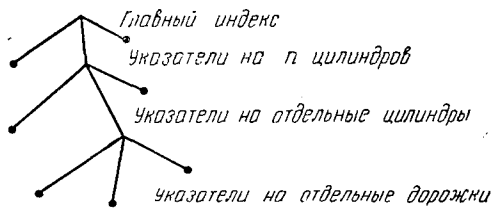


Рис. 17.1. Организация указателей индексно-последовательного файла

Записи файла записываются в основную область данных на томе в виде упорядоченной последовательности по ключам до тех пор, пока эта область не заполнится. Если при добавлении записи в файл для нее нет места на дорожке, в которую она должна быть вписана, чтобы не нарушить упорядоченность по ключу, то эта запись размещается в первое доступное место области переполнения. Область переполнения отводится для размещения тех записей, которые не помещаются на дорожках основной области данных. К записи, размещенной в области переполнения, добавляется поле связи для того, чтобы логически связать эту запись с нужной дорожкой. В элементы индекса дорожек также вносятся соответствующие поправки.

Область переполнения запрашивается программистом в виде

области переполнения цилиндров либо независимой области. Область переполнения цилиндров предназначена для хранения записей, появляющихся в результате переполнения дорожек основной области данного цилиндра. Преимущество использования области переполнения цилиндров — уменьшение времени поиска записи, расположенной в области переполнения. Недостатком такой области является наличие неиспользуемого пространства тома, которое возникает в случае неравномерного добавления записей в файл.

Независимая область переполнения предназначена для хранения всех добавляемых записей независимо от принадлежности дорожки, на которой произошло переполнение, какому-либо цилиндру. Преимущество использования такой области переполнения — уменьшение неиспользуемого пространства тома, выделенного под область переполнения.

Практика показывает, что лучше всего запрашивать области переполнения цилиндров достаточно большими, чтобы в них могло разместиться разумное число дополнительных записей, а независимую область переполнения использовать в том случае, когда заполняются области переполнения цилиндров.

Наличие основной области данных и области переполнения обеспечивает добавление новых записей в файл с последовательной организацией без его перезаписи. При длительном использовании индексно-последовательного файла область переполнения будет заполнена произвольным образом, что приведет к увеличению времени, необходимого для поиска требуемых записей. Поэтому файл необходимо периодически реорганизовать. Частота, с которой следует реорганизовать файл, зависит от частоты использования файла и требований к использованию памяти на томах с прямым доступом и к временным характеристикам обработки файла.

Логическая система управления вводом-выводом предоставляет два способа реорганизации файла. Она собирает статистику о числе областей переполнения цилиндров, числе неиспользуемых дорожек в независимой области переполнения, числе обращений к записям в области переполнения, отличной от первой записи. Эти сведения доступны проблемной программе с помощью специальных макрокоманд.

Первый способ реорганизации файла заключается в том, что он может быть последовательно переписан в другой том с прямым доступом или на магнитную ленту, а затем воссоздан снова в исходной основной области данных. Второй способ осуществляет реорганизацию файла за один проход путем формирования новой основной области данных на другом томе с прямым доступом. В этом случае первоначальная основная область данных уже не может использоваться файлом.

Обработка индексно-последовательного файла выполняется двумя способами: последовательным или прямым. Прямая обработка выполняется с использованием базисного языка доступа. В этом случае для обращения к отдельной записи указывается ее ключ.

Все остальные действия по поиску, чтению или записи выполняются логической системой управления вводом-выводом. Последовательная обработка выполняется с использованием языка доступа с очередями. В этом случае все действия, связанные с поиском, чтением или записью, производятся системой управления вводом-выводом.

Частично-последовательный метод доступа используется для обращения к отдельным элементам системного файла и будет рассмотрен в следующей главе.

Таким образом, базисные методы доступа используются для всех типов организации файлов, тогда как методы доступа с очередями применены только к файлам с последовательной и индексно-последовательной организацией.

17.2. Управляющие таблицы

Для выполнения операций ввода-вывода логическая система управления вводом-выводом обращается к физической системе. При этом, так же как и для выполнения собственных функций по обработке файла, логическая система пользуется управляющими таблицами. Часть таблиц располагается в самой проблемной программе, часть относится к таблицам СУПЕРВИЗОРА ВВОДА-ВЫВОДА. В проблемной программе находятся ТАБЛИЦА ОПРЕДЕЛЕНИЯ ФАЙЛА и ТАБЛИЦА ПАРАМЕТРОВ ФАЙЛОВ.

ТАБЛИЦА ОПРЕДЕЛЕНИЯ ФАЙЛА была рассмотрена нами в § 13.2. Первые сорок байтов этой таблицы образуют БЛОК УПРАВЛЕНИЯ ДАННЫМИ, структура которого не зависит ни от организации файла, ни от способа обработки, ни от типа носителя, на котором расположен файл. Оставшаяся часть существенно зависит от всех этих факторов, и ее необходимо рассматривать для каждого отдельного случая. Там содержатся значения счетчиков обмена, различные маркеры и флаги, поля, связанные с методом доступа, и другая информация, которая формируется для выполнения текущей операции ввода-вывода.

ТАБЛИЦА ПАРАМЕТРОВ ФАЙЛОВ содержится в области таблиц высокой памяти проблемной программы. Она содержит информацию о каждом файле, существующем в проблемной программе. Структура таблицы такова: в первом байте находится число файлов проблемной программы, далее следуют входы, по одному для каждого файла. Структура одного входа в таблицу приведена в табл. 17.1.

В байте 1 указывается тип устройства, на котором располагается файл: медленное устройство, магнитная лента или устройство с прямым доступом.

В байтах 2—8 содержится символическое имя файла. Для системного файла здесь указывается имя основной главы.

В байтах 9—14 для медленных устройств указывается адрес устройства, для магнитных лент и устройств с прямым доступом —

адреса таблиц соответствия физических номеров логическим номерам.

Таблица 17.1

Байты	Содержимое
1	Тип устройства
2—8	Символическое имя файла
9—14	Адрес устройства
15—30	Идентификатор файла
31—65	Текущая характеристика файла

В байтах 15—30 указывается текущий идентификатор файла, расположенного на устройстве с прямым доступом. Для файлов на медленных устройствах эти байты резервируются.

В байтах 31—65 располагается текущая информация о файле. Например, для устройства с прямым доступом в этих байтах указывается логический номер тома, его связь с физическим устройством, число дорожек в томе, тип операции ввода-вывода, информация о метке.

Кроме указанных таблиц для файлов, расположенных на устройствах с прямым доступом, используется БЛОК УПРАВЛЕНИЯ ФАЙЛАМИ. Формирование этих таблиц заканчивается во время выполнения макрокоманды OPEN (ОТКРЫТЬ).

17.3. Связь между логической системой управления вводом-выводом и проблемной программой

Программист использует программы логической системы управления вводом-выводом посредством включения в проблемную программу макрокоманд ввода-вывода. Эти макрокоманды можно разбить на три категории.

1. Макрокоманды определения, которые используются для построения таблицы определения файлов. Примером такой макрокоманды в операционной системе ИБМ/360 является DCB, которая с помощью своих параметров дает логической системе управления вводом-выводом определенную информацию о файле данных. Примерами макрокоманд определения файлов в Системе-4 являются DTFPH, DTFSR, DTFDT, DTFIS, DTFPA. Макрокоманды определения файлов указывают метод доступа к файлу, способы буферизации, размеры буфера, блока, записи, используемый тип устройства и другие характеристики файла. Часть этой информации может быть задана в операторе DD языка управления заданиями.

2. Исполнительные макрокоманды, которые определяют входы в программы логической системы управления вводом-выводом. Исполнительные макрокоманды предназначены для указания и вы-

полнения действий с файлом. Например, прочитать запись или записать ее в файл. Для каждого метода доступа существует несколько исполнительных макрокоманд, из которых обычно одна макрокоманда определяет основной тип действия, а другие определяют действия со специфическими устройствами ввода-вывода. Например, для базисного метода доступа основными макрокомандами являются READ (ЧИТАТЬ) и WRITE (ПИСАТЬ). Так как базисный метод доступа не предполагает автоматической буферизации, то рекомендуется после них ставить макрокоманду WAIT (ЖДАТЬ) для правильного выполнения операций ввода-вывода. Все возможности, предоставляемые этими макрокомандами, задаются с помощью параметров этих макрокоманд.

Макрокоманда READ выбирает блок записей из входного файла, читает его в основную память в область, указанную в макрокоманде. Макрокоманда WRITE определяет место в томе, куда надо разместить блок записей, и записывает этот блок записей на носитель.

Примерами основных исполнительных макрокоманд при методе доступа с очередями являются макрокоманды GET (ВВЕСТИ), PUT (ВЫВЕСТИ), PUTX (ВЫВЕСТИ ИСПРАВЛЕННУЮ ЗАПИСЬ). Так как метод доступа с очередями предполагает автоматическую буферизацию, то после этих макрокоманд не следует использовать макрокоманду WAIT. Все возможности, представляемые этими макрокомандами, задаются с помощью параметров.

Макрокоманда GET планирует заполнение буферов ввода, выделяет записи из блока и управляет процедурами восстановления в случае ошибки при вводе. Для многотомных файлов обеспечивается автоматический переход с одного тома на другой.

Макрокоманда PUT планирует вывод блоков из выходных буферов, заполнение выходных буферов, объединение записей в блоки и выполняет процедуры исправления возникающих при выводе ошибок.

Макрокоманда PUTX используется для исправления некоторого файла данных или для его создания. Она позволяет исправлять, замещать или вставлять новые записи в существующий файл данных.

3. Макрокоманды, которые используются для формирования программ обработки файлов и размещения их в проблемной программе. К ним относятся макрокоманды открытия и закрытия файлов OPEN (ОТКРЫТЬ) и CLOSE (ЗАКРЫТЬ).

Транслированная программа не содержит внутри себя программ обработки файлов, входящих в логическую систему управления вводом-выводом, до тех пор, пока не начнется ее выполнение. Начало выполнения определяется макрокомандой OPEN, по которой происходит завершение формирования управляющих таблиц, генерация требуемых программ обработки файлов, размещение этих программ в проблемной программе. Например, если макрокоманды ввода-вывода определили обращение к медленным устройствам, то будут выбраны только программы обработки файлов на медленных

устройствах. Если были определены макрокоманды обращения к файлам на томах с прямым доступом и медленных устройствах, то будут выбраны программы обработки файлов только на этих устройствах. Структура проблемной программы с точки зрения расположения программ обработки файлов логической системы управления вводом-выводом представлена на рис. 17.2.

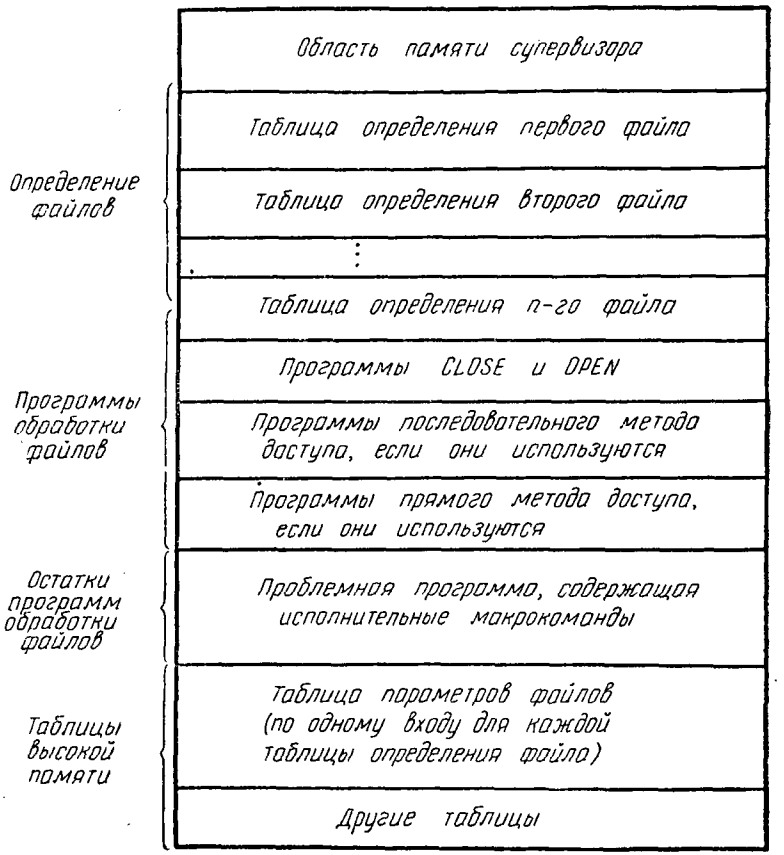


Рис. 17.2. Структура проблемной программы

При использовании индексно-последовательного метода доступа программы обработки файла берутся готовыми из библиотеки макрокоманд ассемблера и вставляются РЕДАКТОРОМ в проблемную программу без дополнительной настройки. Эти программы располагаются в проблемной программе перед таблицами высокой памяти.

Вход в любую программу обработки файлов осуществляется посредством выполнения исполнительной макрокоманды. Внутри области программ OPEN и CLOSE (см. рис. 17.2) имеется таблица переходов, каждый элемент которой представляет собой команду обращения к одной из программ обработки файлов. Исполнительные макрокоманды и осуществляют связь между проблемной программой и системой управления вводом-выводом посредством этой таблицы переходов.

17.4. Макрокоманды формирования программ обработки файлов

Прежде чем произойдет обращение к файлу, должна быть выполнена макрокоманда OPEN. Она обеспечивает доступ к устройству, на котором размещается файл, и выполняет различные проверки. После этой макрокоманды могут следовать исполнительные макрокоманды, обеспечивающие обработку файлов и их ввод-вывод. По окончании работы с файлом должна быть выполнена макрокоманда CLOSE, которая «освобождает» занятое устройство и выполняет действия, связанные с окончанием обработки файла. Макрокоманды OPEN и CLOSE — общие для всех способов организации файлов и методов доступа к ним. Эти команды важнейшие в логической системе управления вводом-выводом. Их назначение — дать возможность проблемной программе обратиться к своим файлам; выполнить ряд проверок, чтобы убедиться, что такое обращение возможно.

Все программы логической системы управления вводом-выводом являются программами *P*1, которые объединяются с проблемной программой посредством макрокоманд ввода-вывода. Программы OPEN и CLOSE состоят как из программ *P*1, так и из программ *P*2, т. е. содержат в себе модули, которые выполняются в состоянии процессора *P*2 и относятся к программам СУПЕРВИЗОРа. Та часть программ CLOSE и OPEN, которая выполняется в состоянии процессора *P*1, объединяется с проблемной программой и располагается в области программ OPEN и CLOSE. Часть программ OPEN и CLOSE, которая выполняется в состоянии процессора *P*2, относится к программам *P*2 СУПЕРВИЗОРа. Поэтому отнесение OPEN и CLOSE к программам обработки файлов является условным.

Макрокоманда OPEN позволяет открыть до шестнадцати файлов одновременно. Она используется для завершения формирования таблицы определения файлов. Посредством параметров в ней могут быть указаны метод обработки, тип файла, информация о действиях, которые надо предпринять при обнаружении конца тома. Файл может быть входным, выходным, а для базисного последовательного метода доступа файл может одновременно являться входным и выходным. Файл может обрабатываться в режиме чтения, записи или модификации. Последовательные файлы на томах магнитной ленты можно читать в обратном направлении. Если па-

раметр метода обработки в макрокоманде опущен, то файл считается входным.

При обнаружении конца тома файл может быть установлен либо в позицию его начала, либо в позицию его конца.

Когда файл открыт, за ним закрепляется устройство, кроме тома с прямым доступом, которое не может быть использовано никаким другим файлом. Том с прямым доступом может быть отведен одновременно несколькими файлами.

Макрокоманда CLOSE позволяет «закрыть» одновременно до шестнадцати файлов. Она используется для того, чтобы прекратить обработку файла, устранить связь между файлом и таблицей определения файла. При этом можно с помощью параметров указать, в какую позицию следует установить файл после его закрытия. После выполнения макрокоманды CLOSE доступ к файлу невозможен до тех пор, пока он заново не будет открыт посредством OPEN.

Если какой-либо файл не был закрыт в проблемной программе, он будет закрыт логической системой управления вводом-выводом по окончании выполнения программы. Результат будет эффективней, если файл закрыт в программе, в которой он обрабатывался.

Программы *P*1 OPEN и CLOSE содержатся в библиотеке загрузочных модулей. Эти программы самостоятельно никогда не выполняются. В область проблемной программы вызываются те из них, которые определены параметрами макрокоманд OPEN и CLOSE. На основе этих параметров определяется также, какие программы *P*1 должны быть резидентными, какие транзитными. Резидентные программы *P*1 OPEN и CLOSE предназначены для определения функций, которые требуется выполнить в каждом конкретном случае, и вызова транзитов. Транзиты *P*1 OPEN и CLOSE предназначены для осуществления различных проверок (например, нет ли ошибок в задании макрокоманд, можно ли выполнять эти макрокоманды, проверить или записать метки файлов на медленных устройствах, перемотать магнитную ленту после использования).

Программы *P*2 OPEN и CLOSE являются транзитными программами СУПЕРВИЗОРа. Они вызываются в основную память в область управляющей программы посредством макрокоманд обращения к СУПЕРВИЗОРу, которые выполняются программами *P*1 OPEN и CLOSE. Программы *P*2 OPEN и CLOSE предназначены для формирования и модификации управляющих таблиц СУПЕРВИЗОРа ВВОДА-ВЫВОДА, например для модификации ТАБЛИЦЫ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ, БЛОКА УПРАВЛЕНИЯ ФАЙЛАМИ. Эти программы устанавливают флаги, открыт файл или закрыт, производят требуемые действия с метками файлов и томов на магнитных лентах и устройствах с прямым доступом, устанавливают магнитную ленту в требуемую позицию.

Порядок вызова транзитов *P*1 и *P*2 макрокоманды OPEN и выполняемые ими действия иллюстрируются рис. 17.3. Рисунок разделен на четыре области: область проблемной программы с мак-

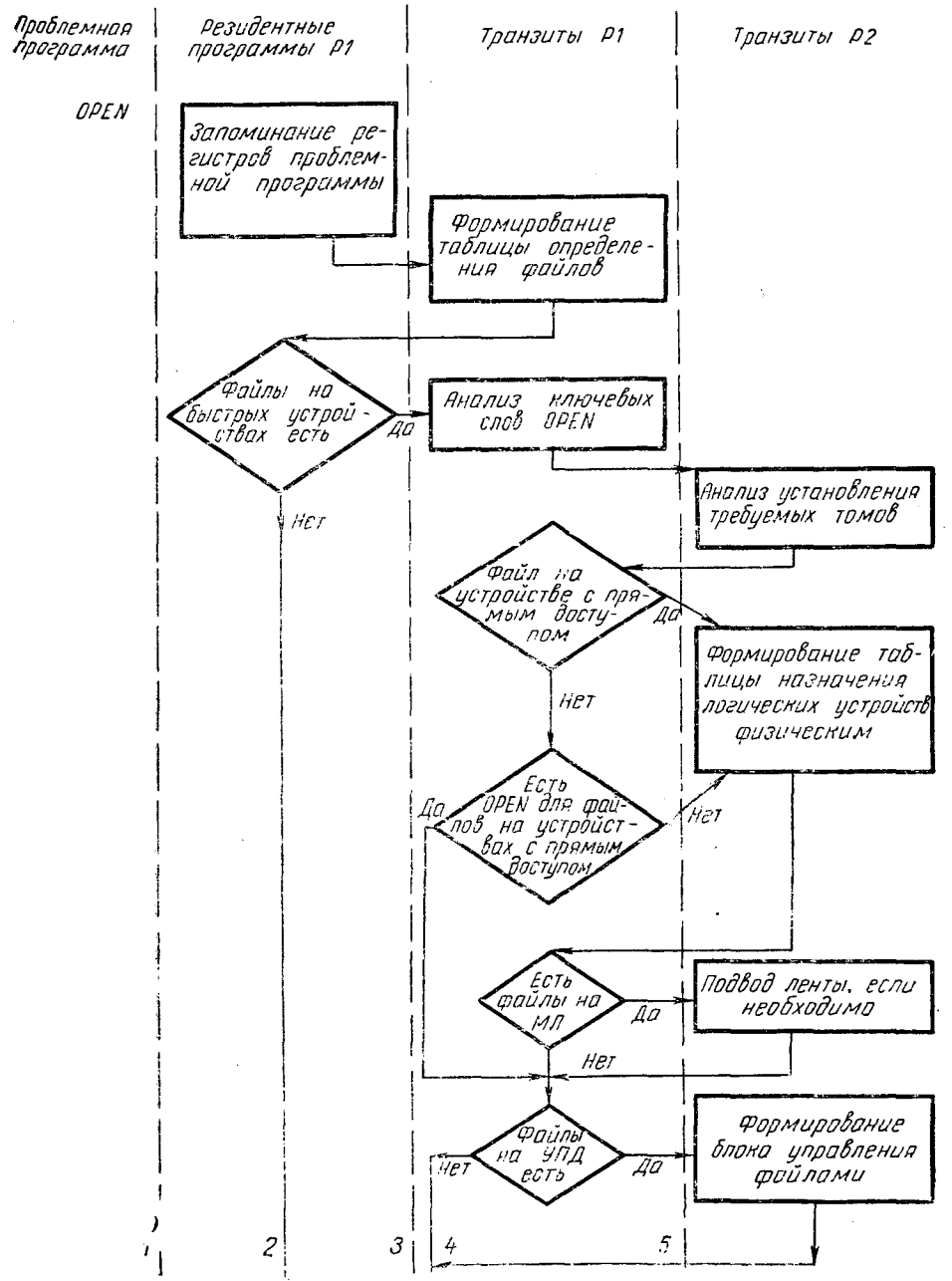


Рис. 17.3. Порядок выполнения макрокоманды OPEN

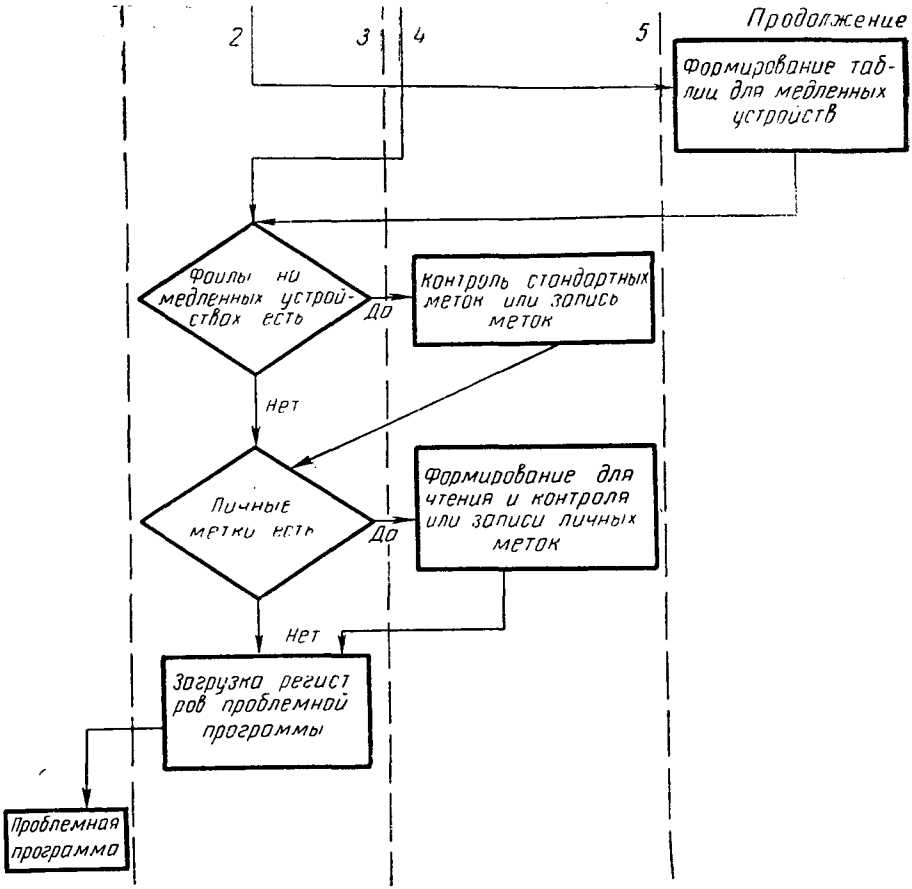


Рис. 17.3. Порядок выполнения макрокоманды OPEN

рокомандами; область, в которой описываются действия резидентных программ *Р*1 макрокоманды OPEN; область, в которой определяются действия транзитов *Р*1 макрокоманды OPEN; область, в которой определяются действия транзитов *Р*2, являющихся программами СУПЕРВИЗОРа. Передача управления между резидентными и транзитными программами *Р*1 макрокоманды OPEN, а также обращение к программам *Р*2 СУПЕРВИЗОРа осуществляется по прерываниям посредством макрокоманд обращения к СУПЕРВИЗОРу.

По макрокоманде OPEN резидентные программы *Р*1 запоминают значение регистров проблемной программы в ее ОБЛАСТИ ПАМЯТИ СУПЕРВИЗОРа и формируют на основании параметров макрокоманды вызов транзитов *Р*1. Транзиты *Р*1 осуществляют проверку правильности задания макрокоманды и производят фор-

мирование ТАБЛИЦЫ ОПРЕДЕЛЕНИЯ ФАЙЛА. После этого управление возвращается резидентным программам *P*1. Резидентные программы анализируют, есть ли уже какие-либо отрытые файлы на магнитных лентах или устройствах с прямым доступом. Если такие файлы есть, то вызываются транзиты *P*1 для анализа ключевых слов макрокоманды OPEN с тем, чтобы правильно вызвать транзиты *P*2. Транзиты *P*2 по метке томов проверяют правильность установления томов, устанавливают соответствующие маркеры и флаги в ТАБЛИЦЕ ОПРЕДЕЛЕНИЯ ФАЙЛОВ и БЛОКЕ УПРАВЛЕНИЯ ФАЙЛАМИ.

Управление возвращается к транзитам *P*1, которые смотрят, где расположен файл. На основании ключевых слов проверяется, есть ли в рассматриваемой макрокоманде файлы на устройствах с прямым доступом. Если есть, то управление передается транзитам *P*2. Если нет, то анализируется, имеются ли другие макрокоманды OPEN для файлов на устройствах с прямым доступом. Если таких файлов нет, то управление также передается транзитам *P*2, которые строят общие части таблиц для быстрых устройств. После этого управление передается транзитам *P*1, которые дополнительно проверяют, есть ли файлы на магнитных лентах, и если есть, передают управление транзитам *P*2. Транзиты *P*2 организуют подвод ленты к началу файла или другой заданной позиции, если необходимо. Управление возвращается транзитам *P*1, которые вновь проверяют, есть ли еще неоткрытые файлы на устройствах с прямым доступом.

Рассмотренные программы строили только общие части таблиц для файлов на магнитных лентах и для файлов на устройствах с прямым доступом. Дополнительная проверка выделяет только файлы на устройствах с прямым доступом, для которых вызываются транзиты *P*2. Транзиты *P*2 заканчивают построение БЛОКА УПРАВЛЕНИЯ ФАЙЛАМИ, проверяют наличие входов типа два и три (см. § 13.2), проверяют метки томов, которые являются расширением основного тома.

Если файлы расположены на медленных устройствах, то резидентные программы *P*1 вызывают транзиты *P*2 для выполнения действий, общих для всех медленных устройств. Осуществляется ряд проверок и производится модификация ТАБЛИЦЫ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ. Управление возвращается резидентным программам *P*1, которые проверяют, есть ли файлы, которые нужно открыть. Если есть неоткрытые файлы на медленных устройствах, то вызываются транзиты *P*1, которые выполняют действия, связанные с контролем и формированием меток файлов на медленных устройствах. Управление возвращается резидентным программам *P*1, которые проверяют, есть ли нестандартные метки. Если нестандартные метки есть, то вызываются транзиты *P*1, обрабатывающие нестандартные метки. После этого управление вновь передается резидентным программам *P*1, которые загружают регистры проблемной программы и передают ей управление.

Мы проследили путь выполнения макрокоманды OPEN. Структура макрокоманды CLOSE аналогична OPEN. Порядок вызова транзитов *Р1* и *Р2* макрокоманды CLOSE иллюстрируется рис. 17.4.

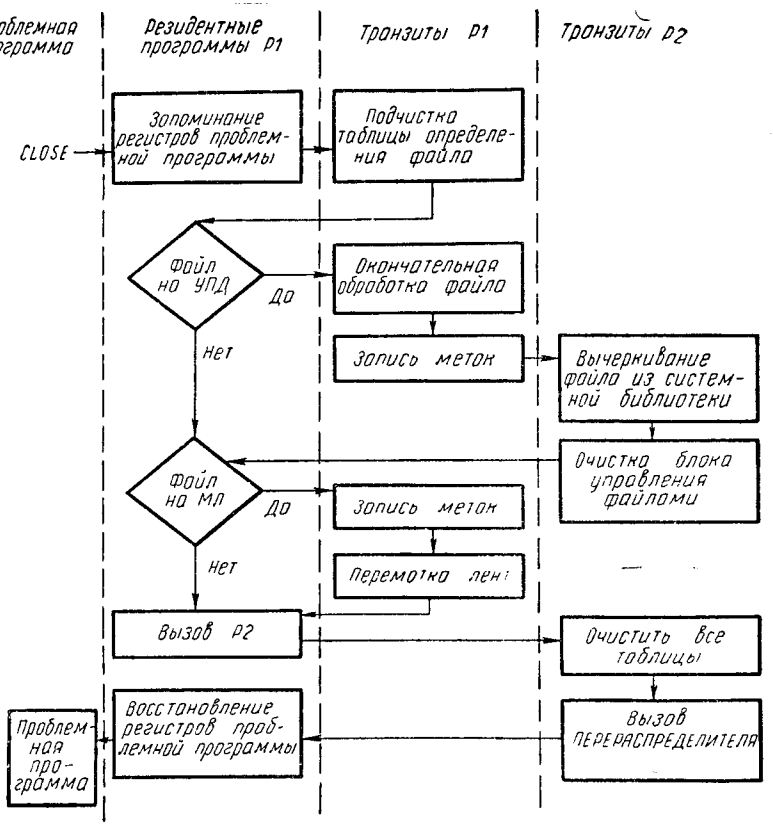


Рис. 17.4. Порядок выполнения программы CLOSE

По макрокоманде CLOSE резидентные программы *Р1* запоминают в ОБЛАСТИ ПАМЯТИ СУПЕРВИЗОРа регистры проблемной программы и вызывают транзитные программы *Р1* проверки правильности задания макрокоманды. В ТАБЛИЦЕ ОПРЕДЕЛЕНИЯ ФАЙЛА устанавливаются соответствующие флаги и маркеры. Управление передается резидентным программам *Р1*, которые анализируют, нужно ли закрывать файлы на устройствах с прямым доступом. Если такие файлы есть, то вызываются транзиты *Р1* и им передается управление. Транзиты *Р1* закрывают файлы, записывают стандартные и нестандартные метки, если необходимо, и вызывают транзиты *Р2*.

Транзиты *P*2, если необходимо, вычеркивают файл из системных библиотек, очищают БЛОК УПРАВЛЕНИЯ ФАЙЛАМИ, освобождают устройства, т. е. подчищают ТАБЛИЦУ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ, и передают управление резидентным программам *P*1. Программы *P*1 проверяют, есть ли файлы на магнитных лентах, и если есть, вызывают транзиты *P*1 для закрытия этих файлов.

Транзиты *P*1 перематывают ленты, записывают стандартные и нестандартные метки и возвращают управление резидентным программам *P*1, которые вызывают транзиты *P*2. Программы *P*2 очищают маркеры и флаги ТАБЛИЦЫ ИНФОРМАЦИИ ОБ УСТРОЙСТВАХ, вычеркивают файлы с устройства, если необходимо, и вызывают ПЕРЕРАСПРЕДЕЛИТЕЛЬ, чтобы указать ему, какие устройства освободились. После этого управление вновь передается резидентным программам *P*1, которые загружают регистры проблемной программы и возвращают ей управление.

17.5. Доступ к системному файлу

Системный файл организован разделами, которые называются основными главами. Каждая основная глава состоит из одной или нескольких вторичных глав. Вторичные главы представляют собой файлы с последовательной организацией. Специфика организации системного файла оказывает влияние на метод поиска отдельного элемента. Поэтому для обращения к системному файлу используется частично-последовательный метод доступа. Физическая и логическая структура системного файла приведена в § 10.2.

Доступ к системному файлу осуществляется на физическом и логическом уровне с помощью специальных макрокоманд. Каждая макрокоманда представляет собой одну или несколько программ обработки системного файла. С функциональной точки зрения набор программ обработки системного файла аналогичен набору программ обработки обычных файлов. Часть программ обработки системного файла относится к программам СУПЕРВИЗОРа, часть является программами *P*1.

Для обращения к системному файлу на физическом уровне используются макрокоманды типа EXDP, являющиеся программами СУПЕРВИЗОРа и рассмотренные нами в § 13.3.

Для обращения к системному файлу на логическом уровне используются два типа макрокоманд: макрокоманды для обращения к основным главам и макрокоманды обращения ко вторичным главам. Макрокоманды обращения к основным главам подготавливают доступ к элементам системного файла. Макрокоманды обращения ко вторичным главам реализуют операции ввода-вывода, т. е. сам доступ к элементам системного файла.

В основу метода доступа к системному файлу положен также метод таблиц. Рассмотрим используемые управляющие таблицы.

Так как обращение к системному файлу аналогично обращению

к обычному файлу, то его необходимо сначала определить, затем открыть, обработать и закрыть. Определение системного файла осуществляется в табличной форме. Формирование таблицы определения для одного элемента системного файла осуществляется с помощью специальных макрокоманд, которые строят ОГЛАВЛЕНИЕ ФАЙЛА. В операционной системе ИБМ/360 такой макрокомандой является BLDL (ПОСТРОИТЬ ОГЛАВЛЕНИЕ), в операционной системе Системы-4 используется макрокоманда DTFPA (ОПРЕДЕЛИТЬ ФАЙЛ ДЛЯ ДОСТУПА ГЛАВАМИ).

Структура ОГЛАВЛЕНИЯ ФАЙЛА приведена в табл. 17.2 ОГЛАВЛЕНИЕ ФАЙЛА располагается в области программ *Р3*. Оно формируется для каждой основной главы. ОГЛАВЛЕНИЮ ФАЙЛА присваивается имя, являющееся именем файла, описанного данным оглавлением. Доступ к основной главе будет осуществляться по имени файла, присвоенного его оглавлению.

Для каждой вторичной главы составляется БЛОК УПРАВЛЕНИЯ ГЛАВАМИ, который формируется в тот момент, когда требуется доступ к данной вторичной главе. БЛОК УПРАВЛЕНИЯ ГЛАВАМИ располагается в области программ *Р3* СУПЕРВИЗОРа. Его структура приведена в табл. 17.3.

Таблица 17.2

Байты	Содержимое
0—5	Имя основной главы
6	Логический номер тома
7	Флаги ошибок
8—11	Адрес цепочки команд канала, если используется EXDP
12—13	Адрес блока управления главой
14—15	Адрес блока управления файлом
16—34	Содержимое регистров канала
35	Флаги СУПЕРВИЗОРа
36—39	Содержимое регистра адреса канала
40—41	Длина таблицы определения файлов
42	Резервируется
43	Монопольное управление основной главой
44—62	Идентификатор вторичной главы
63—65	Флаги ошибок
66—69	Адрес буфера

Все макрокоманды доступа к системному файлу можно разделить на три группы: программы *Р3*, выполняющие перемещение данных; программы *Р2*, управляющие доступом к основной главе; программы *Р2*, управляющие определением текущей позиции во вторичной главе. Все макрокоманды доступа к системному файлу являются программами СУПЕРВИЗОРа.

Таблица 17.3

Байты	Содержимое
0—6	Идентификатор основной главы
7—25	Идентификатор вторичной главы
26—27	Номер предыдущей дорожки *
28—29	Номер следующей дорожки *
30—31	Номер текущей дорожки *
32—37	Абсолютный адрес
38	Тип последней операции (чтение — запись)
39	Номер задачи
40—71	Цепочка команд канала обращения ко вторичной главе
72—79	Список ожидающих задач

* Номера дорожек указываются относительно начала основной главы используются для сокращения времени поиска дорожек, на которых размещена вторичная глава.

Обмен между основной памятью и томом производится по одному сектору в системе ДЖЕЙ и по дорожке в операционной системе ИБМ/360. Дорожки внутри вторичной главы связаны в цепочку. Одна вторичная глава может занимать несколько несмежных дорожек. Чтобы определить цепочку дорожек одной вторичной главы, надо прочитать запись R∅. Затем следует определить адрес требуемой дорожки, после этого следует производить операцию ввода-вывода. Все эти функции выполняются отдельными макрокоманд дами доступа к системному файлу.

Первыми выполняются макрокоманды, строящие ОГЛАВЛЕНИЕ ФАЙЛА. При этом дополнительно создается СПРАВОЧНИК ГЛАВ, который содержит адреса ОГЛАВЛЕНИЙ ФАЙЛОВ всех основных глав.

Следующими должны выполняться макрокоманды, управляющие доступом к главам. Они представляют собой группу программ *P2*, которая аналогична по функциональному назначению макрокомандам OPEN и CLOSE. С их помощью можно открыть файл, который еще не существует и только будет формироваться, или уже существующий. Обращение к файлу производится по имени, которое присвоено ОГЛАВЛЕНИЮ ФАЙЛА. На основании макрокоманд, аналогичных OPEN, производится окончательное формирование ОГЛАВЛЕНИЯ ФАЙЛА и его связь с БЛОКОМ УПРАВЛЕНИЯ ГЛАВАМИ.

Правильное окончание работы этих макрокоманд означает, что в ОГЛАВЛЕНИИ ФАЙЛА устанавливается идентификатор вторичной главы, отличный от идентификатора основной главы. Идентификатор вторичной главы состоит из идентификатора основной главы и идентификатора программы, которая размещается во вторичной главе. Если макрокоманды, управляющие доступом к главе, завершились неуспешно, то в ОГЛАВЛЕНИИ ФАЙЛА уста-

навливается идентификатор вторичной главы, состоящий из идентификатора основной главы, за которым следуют единицы. Такой идентификатор рассматривается как идентификатор основной главы. В этом случае в байтах 63 и 65 ОГЛАВЛЕНИЯ ФАЙЛОВ устанавливаются флаги ошибок.

Макрокоманды, аналогичные CLOSE, предназначены для того, чтобы закрыть главу после ее обработки. Существуют два типа таких макрокоманд. При выполнении макрокоманд первого типа производится вычеркивание идентификатора вторичной главы из ОГЛАВЛЕНИЯ ФАЙЛА. Дорожки, занимаемые данной вторичной главой, освобождаются, перераспределяются в соответствии с нуждами других основных глав. Освобождение дорожек означает установление в нуль битов, соответствующих им, в СПИСКЕ БИТОВ. Основная глава маркируется как «не открытая», а в БЛОКЕ УПРАВЛЕНИЯ ГЛАВАМИ вычеркиваются все задачи, ожидающие доступа к данной главе. Эти задачи маркируются как неожидающие. Задачам сообщается, что вторичная глава, к которой они хотят обратиться, не существует, и если к ней все же необходимо обратиться, то сначала ее следует открыть. В байтах 12—15 ОГЛАВЛЕНИЯ ФАЙЛОВ устанавливаются нули.

При выполнении макрокоманд второго типа производится вычеркивание идентификатора вторичной главы из ОГЛАВЛЕНИЯ ФАЙЛА и установление байтов 12—15 в нуль. Физического разрушения вторичной главы не происходит, т. е. дорожки, занятые вторичной главой, не освобождаются. Если какая-либо задача захочет обратиться к вторичной главе после того, как были выполнены макрокоманды закрытия файла, она должна сначала открыть ее.

Все макрокоманды, управляющие доступом к файлу, работают под управлением программ *Р2* СУПЕРВИЗОРа. Поэтому любая задача, требующая обращения к вторичной главе, которая занята, будет промаркирована как ожидающаяся СУПЕРВИЗОРом. После этих макрокоманд не требуется ставить макрокоманду WAIT (ЖДАТЬ).

Все макрокоманды используют для работы буфер, адрес которого указан в байтах 66—69 ОГЛАВЛЕНИЯ ФАЙЛА.

Программы *Р3*, выполняющие перемещение данных, представляют собой макрокоманды операций ввода-вывода. Они позволяют прочитать или записать вторичную главу в прямом или обратном направлении, модифицировать отдельные секторы вторичной главы. Прочитанный участок вторичной главы может быть записан в то же самое место в томе или в другое.

Макрокоманды ввода-вывода совмещают операции ввода и вывода. Поэтому для своей работы они требуют наличия двух буферов, реализующих обменную буферизацию. В то время как один буфер обрабатывается, второй используется для ввода-вывода. По окончании этих операций буфера меняются ролями: буфер, используемый для обработки, становится буфером ввода-вывода, а буфер ввода-вывода используется для обработки. Адрес второго буфера

указывается в самой макрокоманде. Для правильного выполнения этих операций необходимо после каждой макрокоманды ставить WAIT.

Макрокоманды, реализующие чтение или запись вторичной главы в прямом направлении, позволяют читать или писать следующий сектор дорожки. Если при записи следующего сектора вторичной главы не хватило места на текущей дорожке, то вторичная глава расширяется посредством добавления новой дорожки, на которую и записывается сектор. При этом производится модификация СПИСКА БИТОВ. Если свободных дорожек нет, то устанавливаются флаги ошибок в БЛОКЕ УПРАВЛЕНИЯ ГЛАВАМИ и ОГЛАВЛЕНИИ ФАЙЛА. Флаги ошибок устанавливаются также, если при чтении следующего сектора вторичной главы данного сектора не существует на текущей дорожке или текущая дорожка является последней во вторичной главе.

Если операции ввода-вывода выполнялись для какой-нибудь проблемной задачи и СУПЕРВИЗОРу потребовался доступ к той же вторичной главе, то первым обслуживается запрос СУПЕРВИЗОРа на выполнение операции ввода-вывода. Задача маркируется как ожидающая. Ее запрос будет выполнен после обслуживания запроса СУПЕРВИЗОРа. Макрокоманды, реализующие чтение вторичной главы в обратном направлении, позволяют считать предыдущий сектор дорожки.

При обнаружении начального сектора вторичной главы устанавливается флаг конца обработки файла. Макрокоманды, реализующие запись вторичной главы в обратном направлении, позволяют записать текущий сектор вторичной главы.

Программы *P*2, управляющие определением текущей позиции во вторичной главе, представляют собой макрокоманды, которые дают возможность не последовательно обращаться к вторичной главе, а пропустив несколько секторов в прямом или обратном направлении. Эти макрокоманды позволяют выборочно читать или писать отдельные секторы вторичной главы. Они называются позиционными макрокомандами ввода-вывода и используются в основном для возвращения вторичной главы в исходное состояние, например для повторной операции чтения или записи какого-либо сектора вторичной главы.

Позиционные макрокоманды сами вырабатывают макрокоманду WAIT, поэтому ее не нужно ставить после них. Часть программ *P*2 является резидентными, а часть — транзитами. При составлении этих программ используются все соглашения, рассмотренные в главе 14.

ВОПРОСЫ К ГЛАВЕ

1. В каких случаях эффективна: а) последовательная организация файла; б) прямая?
2. Для чего используется блокирование записей?
3. Какие преимущества получает программист, использующий логический уровень ввода-вывода?

4. Что означает косвенная адресация записей?
5. Чем отличается последовательная и индексно-последовательная организация файлов?
6. Для чего предназначена область переполнения?
7. Типы и назначение макрокоманд ввода-вывода.
8. Назначение макрокоманды: а) OPEN (ОТКРЫТЬ); б) CLOSE (ЗАКРЫТЬ).
9. Структурно из каких программ состоят макрокоманды OPEN (ОТКРЫТЬ) и CLOSE (ЗАКРЫТЬ)?
10. Чем отличается доступ к системному файлу от обычного?

Глава 18

ОБСЛУЖИВАЮЩИЕ ПРОГРАММЫ

18.1. Назначение обслуживающих программ

Обслуживающие программы представляют собой системные программы, предназначенные для обеспечения внутренних потребностей самой операционной системы. В зависимости от своего назначения и выполняемых функций обслуживающие программы можно разделить на пять групп: программы инициации системы, программы поддержания системных библиотек, архив, диагностические программы и самозагружаемые программы.

Программы инициации системы подготавливают формирование и инициацию основных глав, т. е. они предназначены для приема и размещения системного файла на диске. Эти программы выполняют два типа действий:

1) разметку диска, выделив участки для размещения основных глав в соответствии с организацией системного файла и методом доступа к отдельным элементам;

2) создание формата каждой основной главы в соответствии с ее назначением и при необходимости модификацию уже существующих основных глав.

Первый тип действий выполняется программами инициации системного файла. Они записывают дату создания системного диска, формируют оглавление тома. Для каждой дорожки анализируется ее состояние, и если она без дефектов, то для нее формируется и записывается запись P∅. Дефектные дорожки маркируются и пропускаются. На диске также записывается метка, определяющая системный файл. Она используется в дальнейшем СУПЕРВИЗОРом для проверки команд оператора.

Второй тип действий выполняют программы ФОРМИРОВАНИЕ СИСТЕМНОГО ФАЙЛА, НАЧАЛЬНОЕ ФОРМИРОВАНИЕ ОЧЕРЕДИ ЗАДАНИЙ, ИНИЦИАЦИЯ библиотек макрокоманд и стандартных функций.

Программа формирования системного файла создает формат системного файла на диске. На основании параметров управляю-

щих операторов устанавливаются размер системного файла, размер и количество основных глав. На нулевой дорожке формируется УКАЗАТЕЛЬ ПАМЯТИ, который содержит УКАЗАТЕЛЬ ОСНОВНЫХ ГЛАВ и СПИСОК БИТОВ.

Программа начального формирования очереди заданий производит формирование основных глав, которые будут хранить очереди заданий. Ими являются библиотека заданий и библиотека каталогизированных процедур.

Формирование основной главы означает образование указателя вторичных глав и модификацию записей RØ тех дорожек, которые отведены данной основной главе. Посредством записей RØ создается цепочка дорожек, отведенных данной главе.

Очередь заданий будет занимать столько вторичных глав в основной главе с библиотекой заданий, сколько потоков выделено при генерации операционной системы.

В основной главе с библиотекой каталогизированных процедур будут сформированы две вторичные главы. Одна вторичная глава предназначена для хранения каталога заданий, а вторая — справочника каталогизированных процедур.

Программы инициации библиотек макрокоманд и стандартных функций выполняют действия, аналогичные описанным выше, т. е. они формируют указатели вторичных глав и модифицируют записи RØ. Таким образом, создается преформат этих основных глав. Сами библиотеки остаются пустыми. Для наполнения библиотек вызываются программы второй группы. Заполнение библиотеки макрокоманд производит программа поддержания библиотеки макрокоманд, а заполнение библиотеки стандартных функций производит программа поддержания библиотеки стандартных функций. Для этих программ выполнены все подготовительные работы, открыты нужные главы.

Программы поддержания системных библиотек обеспечивают перемещение внутри основной главы, вычеркивание устаревших, добавление новых, замену или модификацию неправильных вторичных глав. Их действия подразделяются на два типа: выполнение перечисленных выше функций по отношению к любой вторичной главе и выполнение специфических функций по отношению к специфическим вторичным главам.

Деление программ поддержания на два типа вызвано тем, что различные вторичные главы имеют различную внутреннюю структуру. При этом некоторые компоненты этих глав зависят от фактического размещения их на диске. Например, вторичные главы, содержащие различные указатели и справочники, хранят абсолютные адреса этих указателей на диске. Поэтому любое перемещение вторичной главы требует перевычисления этих абсолютных адресов.

Первый тип действий выполняется программой ПОДДЕРЖАНИЕ СИСТЕМЫ. Она выполняет свои функции для любой основной главы. С ее помощью можно также содержать в порядке биб-

лиотеки исходных и объектных модулей, входных и выходных данных. Эта программа выполняет следующие функции:

перемещение вторичной главы из одной основной главы в другую, расположенную на другом томе;

вычеркивание вторичной главы из основной главы с соответствующей модификацией списка вторичных глав и указателя памяти;

распечатку указателя вторичных глав любой основной главы.

Вторую группу действий выполняют программы ПОДДЕРЖАНИЕ БИБЛИОТЕКИ ЗАГРУЗОЧНЫХ МОДУЛЕЙ, ПОДДЕРЖАНИЕ БИБЛИОТЕКИ СТАНДАРТНЫХ ФУНКЦИЙ, ПОДДЕРЖАНИЕ БИБЛИОТЕКИ МАКРОКОМАНД, ОБСЛУЖИВАНИЕ МАКРОБИБЛИОТЕКИ.

Программы поддержания библиотеки загрузочных модулей обслуживают все библиотеки, хранящие загрузочные модули, и выполняют следующие функции:

добавление подготовленных заранее на каком-либо томе загрузочных модулей в требуемую библиотеку на системном диске;

вычеркивание ненужных загрузочных модулей из библиотеки;

распечатку списка загрузочных модулей, находящихся в текущий момент в библиотеке;

перемещение отдельных загрузочных модулей из одной библиотеки в другую. Предполагается, что обе библиотеки находятся в одном томе. Такая необходимость возникает, например, если программа, находящаяся во временной библиотеке, отлажена и ее необходимо сохранить для дальнейшего использования. В этом случае отлаженный загрузочный модуль перемещается в постоянную библиотеку загрузочных модулей, временную библиотеку, библиотеку системы подготовки. Формат вторичных глав этих библиотек одинаков.

Для содержания в порядке библиотеки стандартных функций используется программа поддержания этой библиотеки, которая выполняет следующие функции:

добавление новых стандартных функций в библиотеку. При этом новые функции должны быть расположены либо в библиотеке объектных модулей, которая находится в том же томе, либо в библиотеке стандартных функций, которая расположена в другом томе;

вычеркивание ненужных стандартных функций из библиотеки;

распечатку списка стандартных функций, находящихся в библиотеке в текущий момент.

Инициацию этой библиотеки производят программы первой группы.

Для содержания в порядке библиотеки макрокоманд используются две программы: поддержание и обслуживание. Программа поддержания библиотеки макрокоманд выполняет следующие функции:

добавление новых макрокоманд в библиотеку. Новые макрокоманды вводятся с перфокарт;

добавление новых макрокоманд в библиотеку на системном томе

из библиотеки исходных модулей, также находящейся на системном томе;

перемещение макрокоманд из библиотеки, находящейся на любом томе, в библиотеку на системном томе;

вычеркивание ненужных макрокоманд из библиотеки.

Программа обслуживания макробиблиотеки выполняет следующие функции:

распечатку указателя всех макрокоманд, находящихся в библиотеке;

распечатку всех или нескольких макрокоманд;

перемещение макрокоманд из библиотеки макрокоманд в библиотеку исходных модулей, находящуюся на том же самом томе.

Перемещение выполняется, если необходимо модифицировать макрокоманду. СИСТЕМА ПОДГОТОВКИ ПРОГРАММ имеет возможность корректировать основную главу с библиотекой исходных модулей. Размещение макрокоманды в библиотеке исходных модулей позволяет обычным путем откорректировать макрокоманду. Правильная макрокоманда с помощью программы поддержания библиотеки макрокоманд может быть вновь размещена в библиотеке.

Архив представляет собой совокупность программ, позволяющих создать копии всего системного файла или отдельных его компонентов. Копии можно использовать для восстановления системного файла на диске в случае, если последний по каким-либо причинам испортился, а также для хранения части системных библиотек на магнитных лентах, чтобы не переполнять эти библиотеки на дисках. Кроме того, эти программы позволяют создавать частные библиотеки и даже частный системный файл.

Архивом мы обозначаем совокупность программ. Этим же термином будем обозначать и системный файл, хранящийся на магнитной ленте.

К программам, обслуживающим архив системы, относятся АРХИВ СИСТЕМЫ, КОПИРОВАНИЕ, АРХИВ БИБЛИОТЕК ЗАГРУЗОЧНЫХ МОДУЛЕЙ, АРХИВ БИБЛИОТЕК ИСХОДНЫХ И ОБЪЕКТНЫХ МОДУЛЕЙ.

Программа, создающая архив системы, копирует системный файл на магнитную ленту. Эта магнитная лента может использоваться для хранения системного файла, пересылки его в другие организации для восстановления испорченного системного диска, для модификации отдельных компонентов системного файла.

С помощью программы копирования из полученного системного файла на магнитной ленте можно сформировать частный системный файл на диске пользователя. Это выполняется с целью сохранить системный файл на системном диске и дать возможность пользователю производить любые действия, не боясь испортить основной системный файл. При этом логический номер тома частного системного файла изменяется (делается не равным нулю) и соответствует логическому номеру тома диска пользователя. Системные указате-

ли на диске пользователя также изменяются, чтобы указать, что данный диск содержит частный системный файл.

Программа копирования предназначена для получения копии содержимого любого диска, кроме текущего системного диска, на любом другом диске или магнитной ленте, а также для копирования магнитной ленты на любой диск, кроме системного диска.

Программы формирования архива библиотек загрузочных модулей предназначены для копирования этих библиотек с системного диска на магнитную ленту. К таким библиотекам относятся библиотека загрузочных модулей, временная библиотека, библиотека системы подготовки программ. Копирование этих библиотек производится для того, чтобы можно было модифицировать эти библиотеки, создать частные библиотеки, а также чтобы не перегружать системные библиотеки. Если при копировании библиотеки на магнитную ленту оказывается, что на магнитной ленте содержится такая библиотека, то одновременно с копированием производится модификация библиотеки, существующей на магнитной ленте.

С помощью программы формирования частных библиотек загрузочных модулей производится размещение отдельных загрузочных модулей в соответствующую библиотеку на системном или частном диске с магнитной ленты. При этом можно производить замену устаревших загрузочных модулей в библиотеках на системном или частном диске модулями, хранящимися на магнитной ленте.

Программы, создающие архив библиотек исходных и загрузочных модулей, производят копирование этих библиотек с системного файла на магнитную ленту. Если магнитная лента с этими библиотеками существует, то при копировании производится модификация библиотеки, существующей на магнитной ленте. Получение копий этих библиотек выполняется для того, чтобы не происходило переполнение библиотек на системном диске, а также чтобы произвести модификацию этих библиотек.

Посредством копирования полученной на магнитной ленте библиотеки исходных или объектных модулей на диск можно получить частные библиотеки исходных и загрузочных модулей, а также скорректировать системные библиотеки. Копирование библиотек с магнитной ленты на диск производится программами СИСТЕМЫ ПОДГОТОВКИ ПРОГРАММ.

Для более надежного хранения магнитных лент с архивом системы используется метод поколений. Хранится магнитная лента с текущим системным файлом: магнитная лента с изменениями, которые вносились в системный файл последними; магнитная лента с системным файлом, в который последний раз вносились изменения. В случае порчи системного файла на диске его можно восстановить. Магнитные ленты с архивом хранятся всегда в нескольких экземплярах.

Диагностические программы предназначены для редактирования системных файлов, находящихся на дисках и магнит-

ных лентах, а также для анализа состояния системных файлов. Эти программы могут использоваться в следующих случаях:

во время функционирования системы. Например, какая-либо задача закончилась аварийно. Распечатки основной памяти, занимаемой задачей, оказывается недостаточно для анализа причины аварийного окончания. В этом случае может потребоваться распечатка некоторых таблиц СУПЕРВИЗОРа, которую выполняют диагностические программы;

до начала функционирования, чтобы убедиться в правильной работе системы, особенно если перед этим был сбой;

после окончания выполнения ряда заданий с профилактической целью, чтобы убедиться в нормальном функционировании системы.

К диагностическим программам относятся РАСПЕЧАТКА, АНАЛИЗ, РЕДАКТИРОВАНИЕ ФАЙЛОВ, РЕДАКТИРОВАНИЕ ВТОРИЧНЫХ ГЛАВ.

С помощью программы РАСПЕЧАТКА можно вывести СИСТЕМНЫЙ ЖУРНАЛ на устройство печати. Эта программа печатает содержимое основной главы с выводными данными, а также может использоваться для распечатки основной памяти при аварийном окончании задачи.

С помощью программы АНАЛИЗ можно распечатать информацию, необходимую для анализа состояния основных и вторичных глав. Для анализа могут быть выбраны отдельные или все главы. На основании полученной распечатки можно судить о том, не переполнена ли глава, есть ли свободные дорожки для расширения главы, из каких вторичных глав состоит основная глава, можно ли добавить вторичную главу и т. д.

Программа редактирования файла предназначена для подготовки к печати и распечатки отдельных дорожек диска. Дорожки, которые надо распечатать, указываются в управляющих операторах на картах. За один прогон программы можно отредактировать и распечатать отдельные дорожки различных дисков, за исключением текущего системного диска.

Программа редактирования вторичных глав предназначена для подготовки к выводу и распечатки содержимого вторичных глав, находящихся на системном диске. За один прогон программы можно распечатать несколько вторичных глав. Образец печати задается пользователем посредством управляющих операторов на картах.

Самозагружаемые программы предназначены для получения распечаток основной памяти, регистров, файлов на дисках и магнитных лентах. Самозагружаемые программы в какой-то степени дублируют диагностические программы. Основным отличием является то, что все рассмотренные нами группы обслуживающих программ, в том числе и диагностические, работают под управлением СУПЕРВИЗОРа, тогда как самозагружаемые программы выполняются не под управлением СУПЕРВИЗОРа. Они самостоятельно могут выполнить операции ввода-вывода. Загрузка их в

основную память также осуществляется без вмешательства СУПЕРВИЗОРа.

Самозагружаемые программы используются в тех случаях, когда произошло разрушение СУПЕРВИЗОРа, и действие диагностических обслуживающих программ невозможно. Такая специфика ввода самозагружаемых программ в основную память и выполнения их, естественно, усложняет эти программы и требует другого подхода к их проектированию.

К самозагружаемым программам относятся ДИАГНОСТИЧЕСКАЯ РАСПЕЧАТКА, АНАЛИЗАТОР, ПЕЧАТЬ ЛЕНТ, ПЕЧАТЬ ДИСКОВ.

ДИАГНОСТИЧЕСКАЯ РАСПЕЧАТКА используется при возникновении сбойных ситуаций для распечатки регистров и всей основной памяти. Вывод осуществляется либо на устройство печати, либо на магнитную ленту, если по каким-либо причинам устройство печати нельзя использовать.

АНАЛИЗАТОР используется для анализа содержимого оглавления тома. Для этого производится распечатка меток тома, содержимого нулевой дорожки и нулевого цилиндра диска. Управляющие операторы вводятся с консоли. Выдача результатов осуществляется на устройство печати. Печать лент используется для распечатки файлов, находящихся на магнитных лентах. При этом предварительно осуществляется редактирование файлов. Образец печати задается посредством управляющих операторов на картах. Вывод осуществляется на устройство печати.

ПЕЧАТЬ ДИСКОВ используется для редактирования и распечатки содержимого отдельных дорожек диска. Вывод осуществляется либо на устройство печати, либо на магнитную ленту на основании управляющих операторов на картах.

18.2. Принципы построения и использования обслуживающих программ

Обслуживающие программы групп 1—4 выполняются под управлением СУПЕРВИЗОРа. Поэтому метод использования и выполнения этих программ одинаков: стандартизированные сообщения об ошибках и действия, которые необходимо предпринять при их появлении. Для упрощения их использования стандартизированы также используемые ими управляющие операторы и параметры.

В основу проектирования всех обслуживающих программ положен параметрический принцип. Все они написаны с использованием макрокоманд. Наиболее часто повторяемые операции объединены в стандартные программы. Предпринята попытка выделить стандартные программы как внутри одной группы, так и общие для всех групп. Хотя нет единого метода написания обслуживающих программ групп 1—4, метод их использования можно сделать единым. Принципы их построения также просты и похожи.

При проектировании самозагружаемых программ прежде всего необходимо выяснить:

назначение программы;

устройство выдачи информации (устройство печати, магнитная лента или на оба носителя);

методы связи с оператором;

мероприятия по устранению ошибок;

вывод содержимого сверхбыстрой памяти, регистров, основной памяти, которые могут быть испорчены.

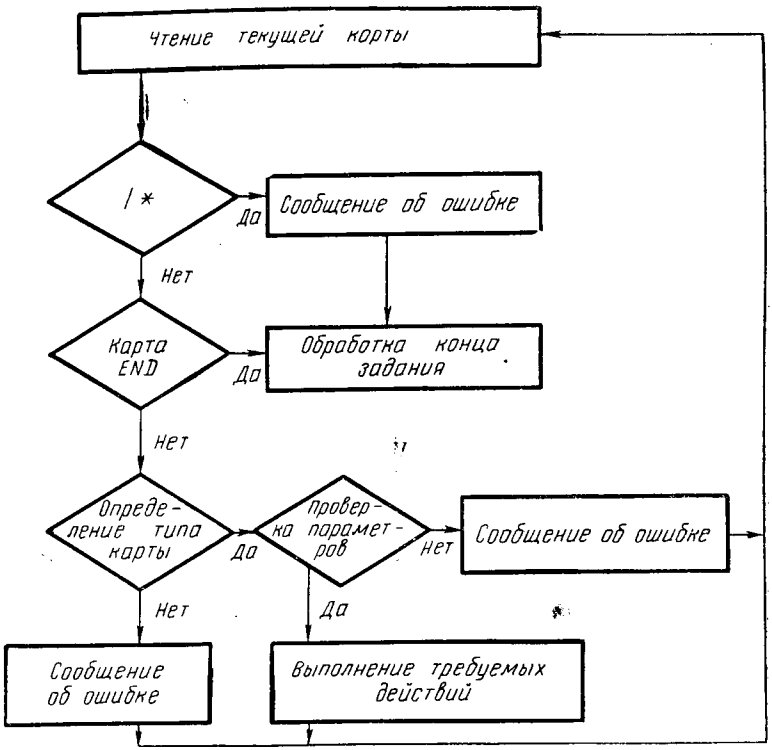


Рис. 18.1. Обработка управляющих карт

Специфика проектирования самозагружаемых программ проявляется в том, что они сами должны предусмотреть выполнение операций ввода-вывода, в то время как для программ групп 1—4 эти операции осуществляются СУПЕРВИЗОРом. Все обслуживающие программы групп 1—4 представляют собой проблемные программы, для которых справедливы все соглашения, принятые для проблемных программ. Самозагружаемые программы имеют иерар-

хическое построение. Они должны выполняться в трех состояниях процессора *Р1*, *Р2*, *Р3*. Поэтому к ним применены все соглашения, принятые при проектировании этих программ.

Рассмотрим более подробно вопросы проектирования обслуживающих программ групп 1—4.

По ходу изложения назначения отдельных программ мы говорили, что требуемые действия определяются параметрами управляющих операторов, которые вводятся с карт. Способ, которым эти карты обрабатываются, стандартизированный. Схема обработки управляющих карт приведена на рис. 18.1, а программа обработки может быть оформлена в виде стандартной.

Назначение программы обработки управляющих карт — распознать тип оператора на карте, определить, правильно ли заданы параметры этого оператора. Если все правильно, то управление передается программе, выполняющей действия, определенные параметрами. Если тип оператора не определен или неправильно указаны параметры, то формируется сообщение оператору об ошибке с указанием ее характера и обрабатывается следующий оператор. Программа обработки управляющих карт аналогична программе ВВОД ЗАДАНИЙ.

Можно выделить еще одну общую программу для всех программ групп 1—4. Такой программой является определение способов выполнения операции ввода-вывода. Управляющие параметры могут вводиться с устройства чтения карт либо из основной главы вводных данных, либо как текущие параметры в момент работы программы, находящиеся в одном из ее рабочих файлов. Вывод сообщений и различные распечатки также могут производиться либо на устройство печати, либо в СИСТЕМНЫЙ ЖУРНАЛ. Принципиальная блок-схема программы определения способов выполнения операции ввода-вывода приведена на рис. 18.2.

Так как мы рассматриваем группу обслуживающих программ, которые выполняются под управлением СУПЕРВИЗОРа, то для них справедливы все основные положения, изученные нами. Структура основной памяти, отведенной задаче, которой подчиняется данная обслуживающая программа, совпадает с той, которая изображена на рис. 15.2.

Вычислив последний адрес, отведенный задаче, легко получить доступ к таблицам высокой памяти. Последнее полное слово содержит адрес таблицы параметров файлов — адрес Д. Первый байт этой таблицы содержит число файлов, определенных в этой задаче. Зная адрес последнего слова памяти, отведенного задаче, легко вычислить и адрес поля Б, в котором находится адрес таблицы параметров текущего прогона. Если адрес Б равен нулю, то управляющие параметры находятся либо в главе вводных данных, либо их нужно вводить с устройства чтения карт. Если адрес Б отличен от нуля, то управляющие параметры являются текущими для программы, файл уже открыт, и для него существует таблица определения файла по адресу, указанному в поле Б.

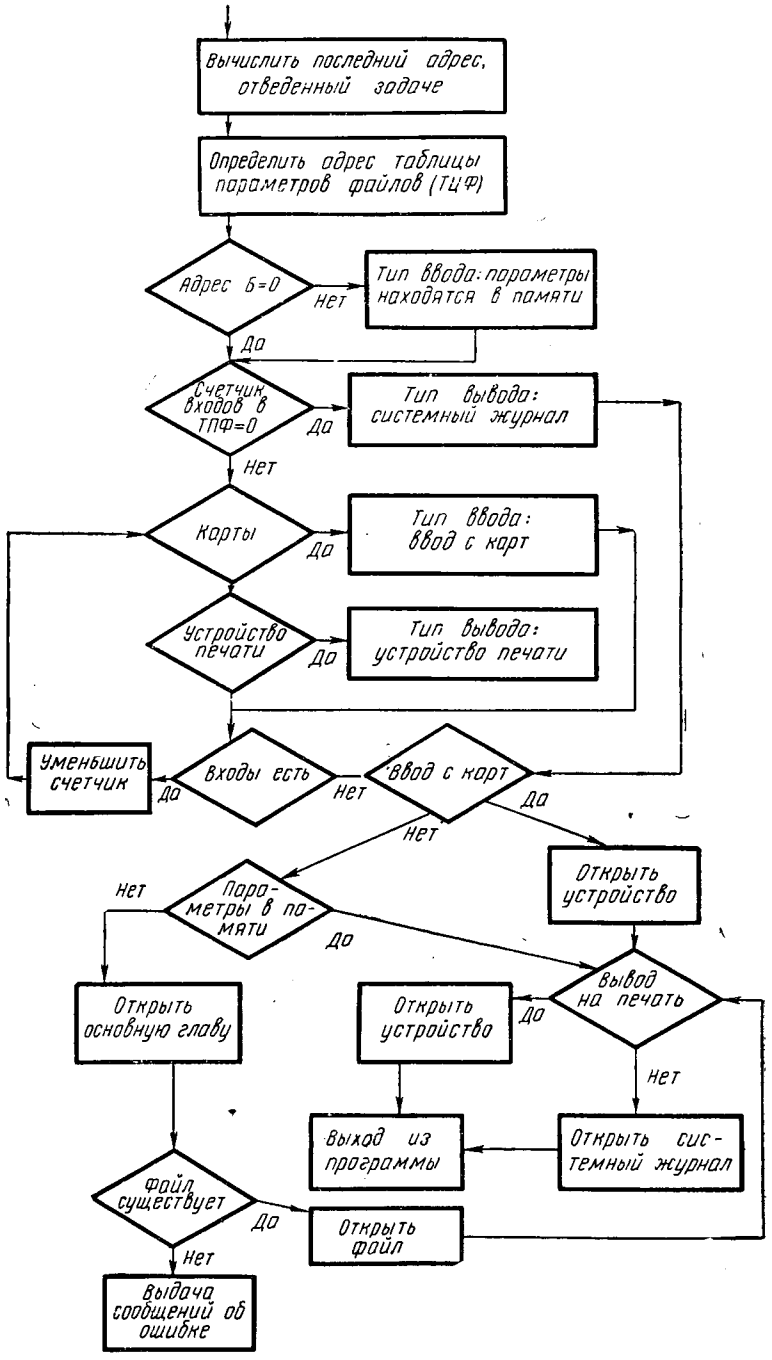


Рис. 18.2. Определение операции ввода-вывода

Если счетчик числа файлов (первый байт таблицы параметров файлов) равен нулю, то все выходные файлы должны записываться в СИСТЕМНЫЙ ЖУРНАЛ. Если счетчик отличен от нуля, то из таблицы параметров файлов выбирается первый вход. В каждом входе таблицы параметров файлов определен тип медленного устройства с указанием производимой операции ввода-вывода (чтение или запись). Это позволяет определить способ выполнения операции ввода-вывода. Для файлов, размещаемых на устройствах ввода с перфокарт или печати, открываются эти устройства. Для файлов, размещенных в СИСТЕМНОМ ЖУРНАЛЕ или в основной главе вводных данных, открываются указанные файлы.

Если управляющие параметры должны вводиться из основной главы вводных данных, но в этой главе нет требуемого файла, то выдается сообщение об ошибке.

В качестве примера, иллюстрирующего основные принципы построения и использования обслуживающих программ, рассмотрим программу поддержания библиотеки каталогизированных процедур. Эта программа относится к программам второй группы — поддержание системных библиотек.

Прежде чем приступить к составлению этой программы, необходимо определить ее назначение и требования к ней. Для этого надо четко знать, что представляет собой библиотека каталогизированных процедур и какой она имеет формат.

Мы знаем, что библиотека каталогизированных процедур содержит описания каталогизированных заданий, что позволяет программисту один раз описать и ввести эти задания, а при повторном использовании во входном потоке заданий указать только имя, присвоенное этой группе заданий. Это значит, что вместо колоды карт, описывающих группу заданий, во входном потоке будет задана одна карта с указанием имени каталогизированной процедуры, но выполняться будет вся группа заданий.

Библиотека каталогизированных процедур размещается в основной главе, которая состоит из трех вторичных глав. Первая вторичная глава содержит указатель вторичных глав, находящихся в данной основной главе. Редактирование этой вторичной главы производится программами первой группы — инициация системы и поэтому в нашем примере не рассматривается.

Следующая вторичная глава содержит каталог заданий. Он представляет собой указатель всех входов в справочнике каталогизированных процедур и содержит для каждого задания его имя, цепочку связей и адрес, по которому хранится описание этого задания в справочнике каталогизированных процедур. Справочник каталогизированных процедур представляет собой вторичную главу и содержит управляющие карты с описанием каждого задания. Управляющие карты хранятся в укороченном формате, т. е. из управляющих карт выброшены все интервалы и ненужная информация.

Зная назначение основной главы и ее формат, можно опреде-

лить назначение обслуживающей программы. Программа должна производить следующие действия:

добавление нового описания каталогизированной процедуры;

замену описания каталогизированной процедуры;

присвоение другого имени каталогизированной процедуре;

объединение двух справочников каталогизированных процедур, находящихся на разных томах;

распечатку содержимого каталога заданий и справочника каталогизированных процедур.

Так как в основу проектирования обслуживающих программ положен парамегрический принцип, то следует определить форматы управляющих карт, чтобы задать все требуемые действия. Для нашего примера можно выделить следующие управляющие карты:

|| РАЗМЕСТИТЬ ТОМ *ппп*

|| РАЗМЕСТИТЬ ТОМ

Карта РАЗМЕСТИТЬ позволяет указать том, на который должна быть записана корректура. Если используется ключевое слово ТОМ с указанием логического номера тома, заданного посредством *ппп*, то корректура будет размещена на указанный том. Если используется ключевое слово ТОМ без указания номера логического тома, то корректура будет размещена на тот же том, на котором расположена библиотека каталогизированных процедур.

Формат управляющих карт выбирается в соответствии с требованиями конкретной операционной системы. В настоящее время принята английская нотация для обозначений на управляющих картах:

|| ВСТАВИТЬ ВСЕ

|| ВСТАВИТЬ <ИДЕНТИФИКАТОР> (*)

|| ВСТАВИТЬ <ИДЕНТИФИКАТОР> <НОМЕР>

Карта ВСТАВИТЬ позволяет добавить новое описание — описание каталогизированной процедуры. Параметры позволяют указать: добавляется одно описание, несколько или весь введенный справочник каталогизированных процедур.

Параметр ВСЕ позволяет указать, что необходимо дописать весь введенный справочник каталогизированных процедур, который определен как КОРРЕКТУРА, к существующему справочнику. Параметр ВСЕ позволяет объединить два справочника, если один из них определить как основной, а второй как корректуру.

Параметр <ИДЕНТИФИКАТОР> <НОМЕР> позволяет вставить введенное описание каталогизированной процедуры. Идентификатор, указанный в скобках Бэкуса, определяет имя введенной процедуры, а номер определяет порядковый номер входа для размещения его в справочнике каталогизированных процедур.

Параметр <ИДЕНТИФИКАТОР> (*) позволяет дописать в существующий справочник группу введенных заданий, обозначенных как корректура. При этом идентификатор введенной группы

описаний заданий должен совпадать с одним из идентификаторов процедуры в справочнике. Если такого идентификатора не найдено, то выдается сообщение об ошибке.

|| ЗАМЕНИТЬ <идентификатор — 1> <идентификатор — 2>

|| ЗАМЕНИТЬ ВСЕ

Карта ЗАМЕНИТЬ используется для модификации существующего справочника каталогизированных процедур.

Параметр <идентификатор —1> и <идентификатор —2> означает, что описание процедуры с идентификатором —1 надо заменить введенным описанием с идентификатором —2. При этом значения идентификатора —1 и идентификатора —2 могут совпадать.

Параметр ВСЕ означает, что весь справочник каталогизированных процедур надо заменить введенной корректурой.

|| ОБЪЕДИНИТЬ

Карта ОБЪЕДИНИТЬ используется для объединения двух справочников, расположенных на разных томах. Полученный справочник каталогизированных процедур будет размещаться на системном томе.

|| ПЕРЕИМЕНОВАТЬ <идентификатор — 1>
<идентификатор — 2>

Карта ПЕРЕИМЕНОВАТЬ позволяет присвоить процедуре, имеющей имя в справочнике идентификатор —1, имя, указанное идентификатором —2.

|| ПЕЧАТЬ *ппп* КАТАЛОГ

|| ПЕЧАТЬ *ппп* СПРАВОЧНИК <идентификатор>

|| ПЕЧАТЬ *ппп* СПРАВОЧНИК ВСЕ

Карта ПЕЧАТЬ используется для указания, какую необходимо произвести распечатку *ппп*, определяет номер тома, с которого производится распечатка.

Ключевое слово КАТАЛОГ определяет, что нужно распечатать вторичную главу, содержащую каталог заданий.

Ключевое слово СПРАВОЧНИК определяет, надо ли распечатать всю вторичную главу со справочником каталогизированных процедур (в этом случае указывается параметр ВСЕ) или надо распечатать одну каталогизированную процедуру (в этом случае указывается ее идентификатор).

После того как определены управляющие операторы, форматы карт и действия, связанные с каждой картой, необходимо определить, с какими файлами будет оперировать обслуживающая программа. Такими файлами будут: файл управляющих операторов, который может вводиться либо с карт, либо находиться в основной главе вводных данных, являющейся в данном случае промежуточным носителем, либо находиться в основной главе каталогизированных процедур; файл сообщений об ошибках и файл распечаток,

которые выдаются на устройство печати или в СИСТЕМНЫЙ ЖУРНАЛ.

Действия обслуживающей программы сводятся к определению способа выполнения операций ввода-вывода и обработке управляющих карт. Эти программы выделены как стандартные. Достаточно составить программы обработки каждой управляющей карты и вставить их в программу обработки управляющих карт, предварительно определив, откуда они вводятся. Блок-схемы этих программ приведены на рис. 18.1 и 18.2.

ВОПРОСЫ К ГЛАВЕ

1. Назовите основные группы обслуживающих программ.
2. Типы действия программ инициирования системы.
3. Основные функции программ поддержания системных библиотек.
4. В каких случаях используются диагностические программы?
5. Чем отличаются самозагружаемые программы от других обслуживающих программ?
6. Основные принципы построения обслуживающих программ.

ЛИТЕРАТУРА

Вычислительная техника. Обработка информации. Словарь терминов. М., Изд-во Комитета стандартов, мер и измерительных приборов при Совете Министров СССР, 1970.

Джермейн К. Программирование на IBM/360. М., «Мир», 1971.

OS/360. Супервизор и управление данными. М., «Советское радио», 1972.

Липаев В. В., Колин К. К., Серебровский Л. А. Математическое обеспечение управляющих ЦВМ. М., «Советское радио», 1972.

Супервизоры и операционные системы. М., «Мир», 1972.

Функциональная структура OS/360. М., «Советское радио», 1971.

Супервизор и управление данными. М., «Мир», 1972.

РАЗДЕЛ 5

ПАКЕТЫ ПРИКЛАДНЫХ ПРОГРАММ

Глава 19

ОСНОВНЫЕ ПОНЯТИЯ ПАКЕТОВ ПРОГРАММ

19.1. Назначение пакетов программ и требования к ним

Пакет прикладных программ представляет собой комплекс программных средств, рассчитанных на некоторый класс задач и определенного потребителя. Пакеты прикладных программа предназначены: для решения типовых научно-технических, инженерных, экономических и специальных задач; обеспечения работы мультипроцессорных и многомашинных систем; работ в различных режимах; преемственности программ, разработанных для машин второго поколения.

В настоящее время принята следующая классификация пакетов:

I группа — пакеты программ, расширяющие возможности основных операционных систем;

II группа — пакеты прикладных программ общего назначения, обеспечивающие решение типовых научных, инженерных, экономических и специальных задач.

К первой группе относятся пакеты, обеспечивающие работу мультипроцессорных и многомашинных систем, работу в различных режимах (в реальном масштабе времени, разделения времени). Особенностью этих пакетов является то, что при их создании необходимо проектировать СУПЕРВИЗОР, управляющий работой этих пакетов. При этом требуется переделка СУПЕРВИЗОРа основной операционной системы. Реализуется это путем замены отдельных блоков супервизора операционной системы и добавлением новых программ и таблиц. Поэтому пакеты I группы проектируются в соответствии с требованиями и соглашениями, принятыми для операционных систем.

Пакеты программ II группы предназначены для применения электронных вычислительных машин во всех сферах человеческой деятельности. Структура специального программного обеспечения приведена на рис. 19.1.

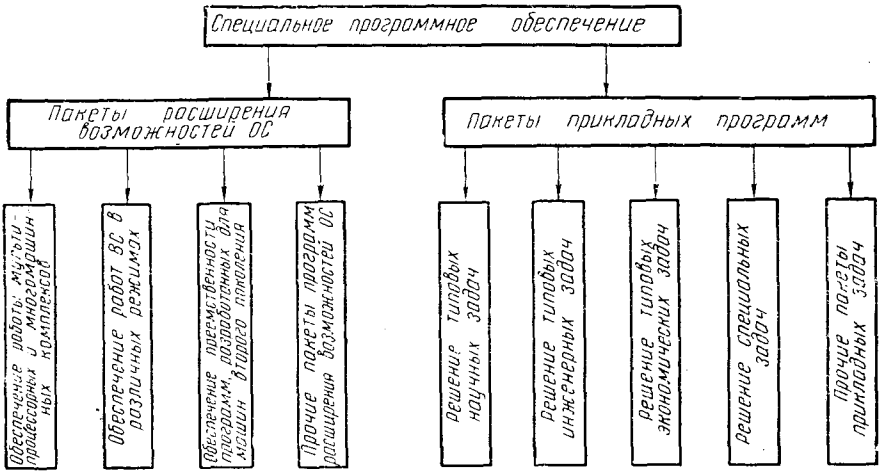


Рис. 19.1. Структура специального программного обеспечения

19.2. Состав пакета программ

Пакет состоит из комплекса программ и документации. Совокупность используемых программных средств внутреннего управления и обслуживания пакета определяет уровень системной организации пакета. Пакет, предназначенный для решения определенного класса задач, может допускать выделение разных уровней системной организации для обеспечения удобств или возможности его использования на различных конфигурациях машин.

С точки зрения организационной структуры пакеты подразделяются на пакеты с простой структурой и пакеты со сложной структурой. Пакеты простой структуры могут иметь две разновидности: как набор независимых программ или как набор взаимосвязанных программ.

При первом способе организации пакеты представляют собой набор программ по типу стандартной библиотеки подпрограмм. Такие пакеты могут содержать любое количество программ. Так, пакет научных программ (SSP) для решения статистических и математических задач для системы ИБМ/360 содержит более 250 программ. Число программ в пакете может непрерывно пополняться.

Программы в пакетах такой структуры осуществляют чисто вычислительные функции и не содержат каких-либо ссылок на устройства ввода-вывода. Пользователь должен дополнять свою программу операторами ввода-вывода, а также соответствующими операторами, необходимыми для полного решения задачи.

Сложные вычислительные процессы могут быть представлены в пакете в виде серии или последовательности программ. Выполнение программ пакета может производиться либо в порядке их рас-

положения, либо в произвольном порядке. В обоих случаях обращение к программам пакета осуществляется с помощью соответствующего оператора обращения к подпрограмме или оператора обращения к процедуре, в зависимости от способа представления пакета.

В общем случае программы пакета могут быть представлены на различных языках. При обращении к ним имя в соответствующем операторе обращения должно соответствовать имени программы в пакете. В программу должны быть включены указания по выполнению трансляции.

Пакет сложной структуры включает: ведущую программу; процессор с входного языка; набор программных модулей, составляющих тело пакета; набор обслуживающих программ.

Ведущая программа предназначена для управления общим ходом работы программ пакета. Она определяет порядок работы программ пакета, формирует данные, необходимые операционной системе для вызова той или иной фазы пакета и передачи ей управления.

Процессор предназначен для обработки программы, написанной на входном языке. Он может представлять собой интерпретатор, компилятор или генератор макрорасширений.

Тело пакета состоит из набора программных модулей, предназначенных для решения определенного класса задач. Каждый программный модуль представляет собой программу или процедуру, реализующую алгоритм вычислительного или логического характера.

Набор обслуживающих программ состоит из ряда программ, выполняющих вспомогательные функции и предоставляющих пользователю определенный сервис при работе с пакетом: средства генерации, редактирования библиотеки, удобство отладки и диагностики ошибок.

Все программы пакета объединяются в единую систему программ, имеющую либо структуру с запланированным перекрытием, либо динамическую структуру. При организации с запланированным перекрытием в памяти машины постоянно находится одна основная часть ведущей программы, называемая корневым сегментом. Все другие подпрограммы загружаются в память и выполняются последовательно по мере необходимости.

Структуры с запланированным перекрытием предназначены для выполнения больших и сложных программ, которые по своим размерам превышают имеющийся объем оперативной памяти.

Общую схему загрузки программ в случае структуры с запланированным перекрытием можно представить следующим образом. В память загружается корневой сегмент ведущей программы. В оставшуюся часть памяти загружаются программы, вызываемые ведущей программой. Таким образом, все выполняемые программы помещаются в память в одно и то же место. При загрузке в память очередной программы для выполнения ранее находившиеся в памя-

ти программы уничтожаются. Они могут быть введены вновь по вызову ведущей программы.

Общая схема взаимосвязи программ в пакете с динамической структурой приведена на рис. 19.2.

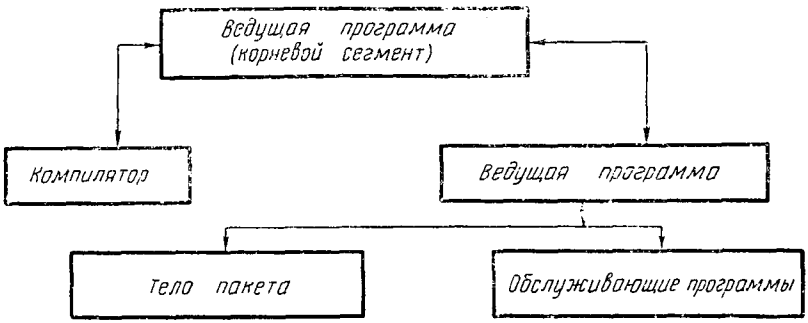


Рис. 19.2. Схема взаимосвязи программ пакета, имеющего сложную структуру

При разработке пакетов сложной структуры могут быть выделены следующие уровни языков:

языки, используемые для написания программ пакета;

входной язык пакета, предназначенный для задания параметров и управляющей информации при настройке пакета на решение конкретной задачи;

язык управления заданиями, предназначенный для ввода программы пользователя в систему и ее реализации.

Структура пакета должна удовлетворять следующим требованиям:

общность и гибкость пакета с целью обеспечения возможности решения широкого круга задач заданного класса;

совместимость прикладных программ пакета с целью обеспечения возможности решения сложных задач заданного класса;

эффективность работы отдельных модулей пакета и всего пакета в целом;

простота и надежность использования пакета с учетом возможного автоматического выбора метода решения задачи;

возможность быстрого получения ответа в форме, требуемой потребителем;

индикация процесса решения задачи;

возможность использования вычислительных модулей одного пакета в другом пакете;

возможность работы в диалоговом режиме.

В процессе разработки пакета оформляется комплект соответствующей документации, который включает комплект рабочей документации и комплект справочной документации.

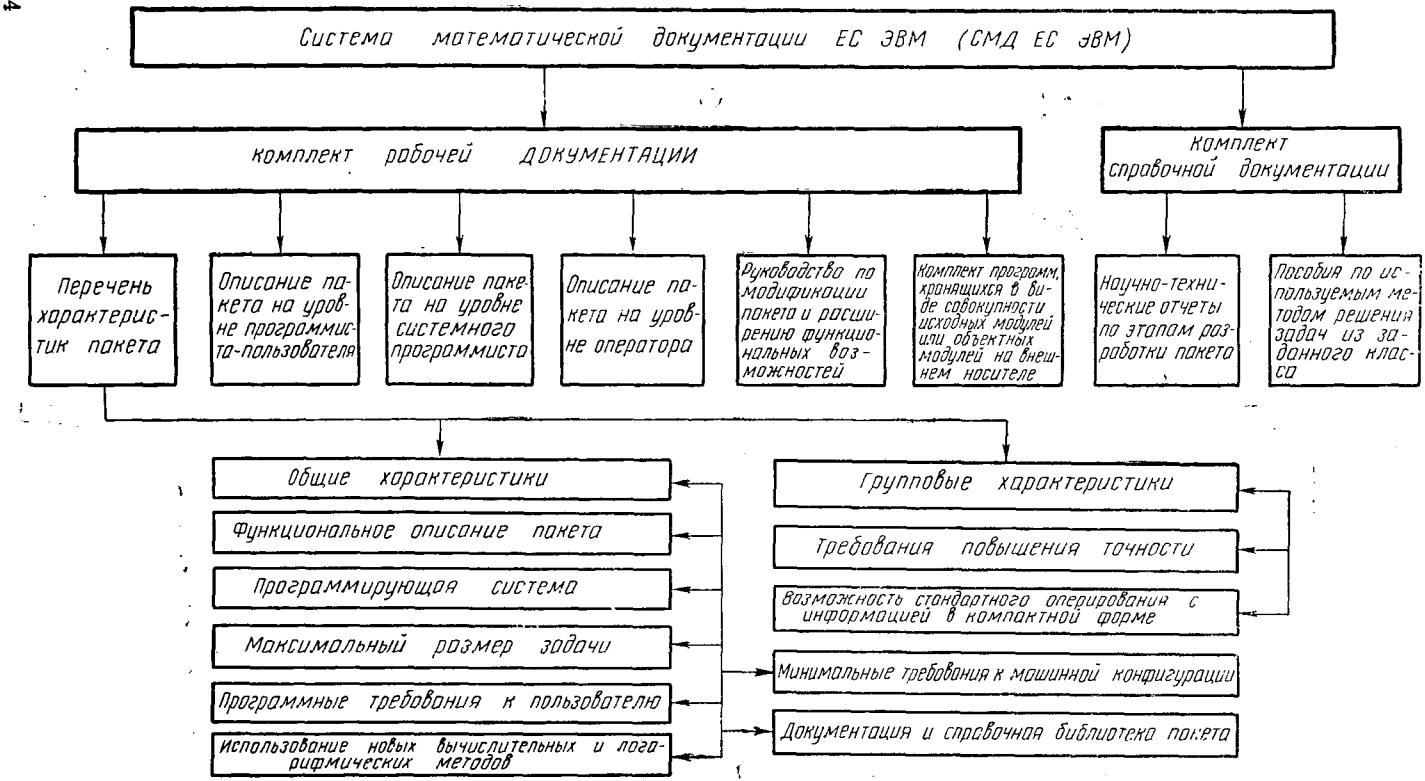


Рис. 19.3. Состав документации ЕС ЭВМ

Комплект рабочей документации включает: перечень характеристик пакета, присущих как всем модулям пакета, так и отдельным группам модулей пакета; описание пакета на уровне пользователя; описание пакета на уровне системного программиста; описание пакета на уровне оператора; руководство по модификации и расширению пакета; компоненты программ пакета.

Комплект справочной документации включает научно-технические отчеты по этапам разработки пакета и пособия по используемым методам решения задач из заданного класса.

Состав комплекта документации пакета ЕС ЭВМ приведен на рис. 19.3.

19.3 Технические требования

Пакеты функционируют под управлением основных операционных систем и должны удовлетворять стандартным соглашениям, предъявленным этими системами к программам пользователя. К ним относятся соглашения о функциональном распределении регистров, о передаче информации от одной программы к другой, о правилах образования задач и их приоритетах, соглашения об использовании ресурсов, о связи с оператором, о ведении системного журнала, об управлении аварийными ситуациями, о способах доступа и управлении данными.

Пакеты прикладных программ должны иметь модульную структуру, допускающую модификацию, замену отдельных модулей и их дальнейшее развитие. Для того чтобы решать одну и ту же задачу с большей производительностью на старшей модели ЭВМ без изменения имеющегося задела программ, пакет должен иметь иерархическую структуру. На нижнем уровне, рассчитанном на минимальную конфигурацию, должны иметься автономные модули по различным типовым операциям. На верхнем уровне, рассчитанном на высокопроизводительные машины, должны быть модули типовых процессов обработки. Используя иерархическое построение пакетов, можно для любой конфигурации машины генерировать решение с использованием минимально необходимого набора модулей пакета.

Входной язык пакета должен быть наглядным, доступным, учитывающим исходные языки операционных систем, а также специфику решаемого класса задач.

Документация пакета прикладных программ должна содержать описания и руководства, необходимые для эксплуатации, редактирования и расширения пакета.

В техническом задании на разработку конкретного пакета должны быть указаны: необходимые системы программирования; вспомогательные программы; минимальная конфигурация машины; требуемые устройства; набор используемых инструкций; минимальный объем памяти; способ хранения пакета на внешних носителях.

19.4. Порядок проведения и приемки работ

Работы по созданию пакета должны предусматривать следующие основные этапы: предварительные исследования, алгоритмизацию исследуемых процессов; программирование; комплексную отладку программ и разработку документации; внедрение.

Цель предварительного исследования — подготовка материалов к разработке проекта пакета. На этом этапе производится сбор материала и его исследование. Этап заканчивается составлением и утверждением технического задания на разработку пакета.

На втором этапе производится алгоритмизация типовых процессов решения заданного класса задач, обоснование выбора применяемых методов, учет особенностей конкретной операционной системы. Этап заканчивается составлением блок-схемы и утверждением входного языка пакета.

На этапе программирования создаются процессор с входного языка, программы пакета и производится их автономная отладка. Этап заканчивается составлением документов на отлаженные модули.

На этапе комплексной отладки пакета программ происходит окончательное уточнение логико-функциональной структуры пакета в целом. Этап заканчивается составлением комплекса документации.

На последнем этапе производится опытная эксплуатация пакета прикладных программ в течение установленного срока.

ВОПРОСЫ К ГЛАВЕ

1. Что понимается под пакетом программ?
2. Назначение пакета.
3. Состав пакета.
4. Характеристика пакета простой структуры.
5. Характеристика пакета сложной структуры.
6. Принципы организации программ в пакете и их краткая характеристика.
7. Уровни языков, используемых в пакетах сложной структуры.
8. Требования, предъявляемые к структуре пакета.
9. Состав документации пакета.
10. Технические требования к пакету.
11. Этапы разработки пакета.

Глава 20

ПАКЕТЫ ПРОСТОЙ СТРУКТУРЫ

20.1. Метод сетевого планирования и управления

В качестве примера пакета простой структуры с взаимосвязанным комплексом программ рассмотрим пакет реализации системы ПЕРТ, предназначенный для проведения временных расчетов по сетям.

Основой метода сетевого планирования и управления является схема, представляющая собой графическое изображение плана. На схеме в определенном порядке располагаются действия, которые должны быть осуществлены для того, чтобы достичь определенной цели.

Графически схема (сеть) изображается с помощью кружочков или прямоугольников, обозначающих события, и линий, соединяющих два события и обозначающих действие или операцию. Событие является контрольной точкой в плане. Оно занимает лишь один момент во времени и обозначает начало или конец какого-нибудь действия. Действие — это обозначение работы в процессе выполнения программы, затрачиваемых ресурсов, затрачиваемых человекочасов. На действие затрачивается время, событие же — это точка во времени.

Действия изображаются в виде линии, соединяющей события. Линия должна иметь стрелку, показывающую направление хода работ. Длина линии действия никогда не означает количества времени или объема ресурсов, потребляемых действием. Действие может также означать зависимость одного события от другого и в таком случае не означает потребление ресурсов. Такие действия называются «фиктивными» действиями, или «фиктивной» работой, и обозначаются пунктирной линией.

Каждая сеть составляется для определенного состава работников, определенного уровня руководства и отражает степень детализации и сферу деятельности, относящуюся к данному уровню руководства. Необходимо четко представлять цель данного плана. Она должна быть отражена в составлении перечня событий и последовательности их выполнения.

Составление сети осуществляется или с начального события в направлении к целевому событию, или в обратном порядке — от целевого события к начальному. Когда все события и действия расположены в правильной последовательности и показаны все необходимые зависимости, для каждого события экспертами определяются три оценки времени: оптимистическая, наиболее вероятная и пессимистическая. Оптимистическая оценка (a) — это минимально необходимое время выполнения операции при благоприятных условиях. Наиболее вероятная оценка (m) — время, которое чаще всего будет соответствовать действительности. Пессимистическая оценка (b) — максимальное время выполнения операции при крайне неблагоприятных условиях. Оценки задаются в одних и тех же единицах измерения: месяцах, днях или часах, и остаются неизменными, если не изменяется объем работы.

На основании временных оценок производится расчет ожидаемого времени операции t_e по формуле

$$t_e = \frac{a + 4m + b}{6}. \qquad (20.1)$$

Например, если три оценки времени равны соответственно 4, 8 и

12 неделям, то $t_e=8$ неделям. Смысл ожидаемого времени операции заключается в том, что статистическая вероятность выполнения действия не позднее чем через восемь недель составляет 50%. Величина t_e — основная для всех временных расчетов в системе сетевого планирования и управления.

Для характеристики неопределенности, присущей оценкам времени, затрачиваемого на каждую операцию, рассчитывается величина дисперсии σ^2. Чем больше разница между оптимистической и пессимистической оценками, тем большая степень неопределенности момента выполнения работы. И наоборот, чем меньше разброс между оптимистической и пессимистической оценками, тем более достоверно выполнение работы в ожидаемый срок.

$$\sigma^2=\left(\frac{b-a}{6}\right)^2. \tag{20.2}$$

Следующим шагом при расчете сети является определение наименьшего ожидаемого времени события T_E и дисперсии относительно этого времени $\sigma^2_{T_E}$:

$$T_{E_i}=\max\left(T_{E_{i-1}}+t_{e_{(i-1,\ i)}}\right). \tag{20.3}$$

Расчет начинается от первоначального события. T_{E_1} будет равно нулю, поскольку началу работ не предшествует ни одна операция, на которую требовалась бы затрата времени.

Расчет дисперсии относительно ожидаемого времени события аналогичен расчету ожидаемого времени события. Величины дисперсии относительно ожидаемого времени действия суммируются по самому длительному пути, ведущему к каждому событию. Расчет дисперсии относительно ожидаемого времени события может быть осуществлен одновременно с расчетом наименьшего ожидаемого времени события.

Затем производится расчет наибольшего допустимого времени события T_L и дисперсии относительно этого времени $\sigma^2_{T_L}$:

$$T_{L_i}=\min\left(T_{L_{i+1}}-t_{e_{(i,\ i+1)}}\right). \tag{20.4}$$

Оценка наибольшего допустимого времени, за которое событие должно быть достигнуто, определяется путем назначения даты свершения целевого или конечного события, а затем проведения вычислений в направлении к более ранним событиям. Дата свершения целевого события может быть установлена равной дате, соответствующей наименьшему ожидаемому времени целевого события. Тогда для целевого события $T_L=T_E$. Дата свершения целевого события может быть задана в виде директивного срока T_s. В этом случае $T_L=T_s$ для целевого события.

При определении дисперсии относительно наибольшего допустимого времени события расчет начинается с целевого события. Поскольку T_L целевого события установлено, то дисперсия относительно наибольшего допустимого времени этого события равна нулю. Величина дисперсии для остальных событий рассчитывается

путем суммирования величин дисперсии относительно ожидаемого времени действий (t_e) вдоль самого длительного пути к каждому событию в обратном направлении.

На основании наименьшего и наибольшего допустимого времени события определяется резерв времени ΔT.

$$\Delta T = T_L - T_E. \tag{20.5}$$

Положительный резерв времени указывает на возможность выполнения работы раньше установленного срока. Нулевой резерв времени указывает на возможность осуществления работы точно в срок, если не будет непредвиденных срывов по другим видам работ. Отрицательный резерв времени указывает на невозможность осуществления работ в установленные сроки.

Если провести линию, соединяющую последовательно все события, имеющие нулевой резерв времени, то она образует путь, соединяющий начальное и целевое события. Путь от начального до целевого события, проходящий через события с наименьшим резервом времени, называется «критическим путем».

Если известны оценки ожидаемого времени многочисленных и разнообразных действий программы, то можно оценить вероятность завершения события в любой будущий момент времени. Оценка вероятности выполнения события в директивный срок определяется по формуле

$$z = \frac{T_s - T_E}{\sqrt{\Sigma \sigma^2_{T_E}}}, \tag{20.6}$$

где z — аргумент нормальной функции распределения вероятностей:

$$P_R = f(z). \tag{20.7}$$

При расчете аргумента z в знаменателе берутся дисперсии для тех событий, которые расположены на критическом пути. После того как z определено, по графику данной функции или по таблице нормальной функции распределения вероятностей и по величине z определяется значение вероятности в процентах.

20.2 Краткая характеристика пакета

Пакет программ обеспечивает пользователю получение сообщений об ошибках и распечатки изменений; распечатки всей сети и распечатки результатов. При распечатке результатов можно получать отчеты, ориентированные на события, или отчеты, ориентированные на операции. Такие отчеты могут содержать:

а) по событиям: наиболее раннюю дату события; наиболее позднюю дату события; резервное время события; директивный срок; вероятность наступления события; критический предшественник — предыдущее событие на критическом пути в направлении к начальному событию; идентификатор события; описание события; дополнительную информацию пользователя;

б) по операциям: наиболее раннюю дату начала; наиболее позднюю дату начала; наиболее раннюю дату окончания; наиболее позднюю дату окончания; общий резерв операции — общее количество времени, на которое можно задержать операцию, не задерживая при этом даты завершения проекта; среднее квадратичное отклонение и вероятность; дополнительную информацию пользователя.

При работе пакета предполагается, что исходные сети представлены в виде библиотеки сетей. Каждая сеть в библиотечном массиве должна иметь сотоветствующий идентификатор.

Сети могут быть ориентированными на события или на операции. В сети, ориентированной на события, должно содержаться полное описание подлежащей достижению цели. В сетях, ориентированных на операции, должно содержаться полное описание работы, подлежащей выполнению между двумя событиями.

В библиотеке сетей каждая операция сети должна иметь следующие характеристики: идентификатор начального события, идентификатор конечного события, длительность операции или дату завершения, описание ключей и дополнительные поля для сортировки. Дополнительные поля сортировки могут использоваться для выделения фиктивных операций.

Сети, находящиеся в библиотеке сетей, могут перед обработкой соответствующим образом корректироваться. Различают три вида корректировки:

1) к библиотеке сетей добавляется новая сеть;
2) изменяется сеть, находящаяся в библиотеке;
3) из библиотеки сетей удаляется некоторая сеть.

В зависимости от этого при обработке сетей различают три типа прогонов: новый прогон, повторный прогон, вычеркивающий прогон. Каждый прогон имеет соответствующий идентификатор.

Для проведения сетевых расчетов с использованием данного пакета пользователь должен ввести данные о сети и значения соответствующих параметров, определяющих режим работы пакета в условиях конкретной системы. К таким параметрам могут относиться: тип прогона, система дат, количество временных оценок на одну операцию, система отчетности, размер описаний и другая информация.

Минимальный комплект устройств, необходимый для пользования данным пакетом, должен включать: основную память емкостью 64 К; одно считавающее устройство с перфокарт или одно считывающее устройство с перфоленты; одно широкоформатное печатающее устройство; один диск или одну магнитную ленту.

20.3. Описание программ пакета

Пакет системы ПЕРТ состоит из десяти логически связанных программ:

1) ввод и проверка данных;

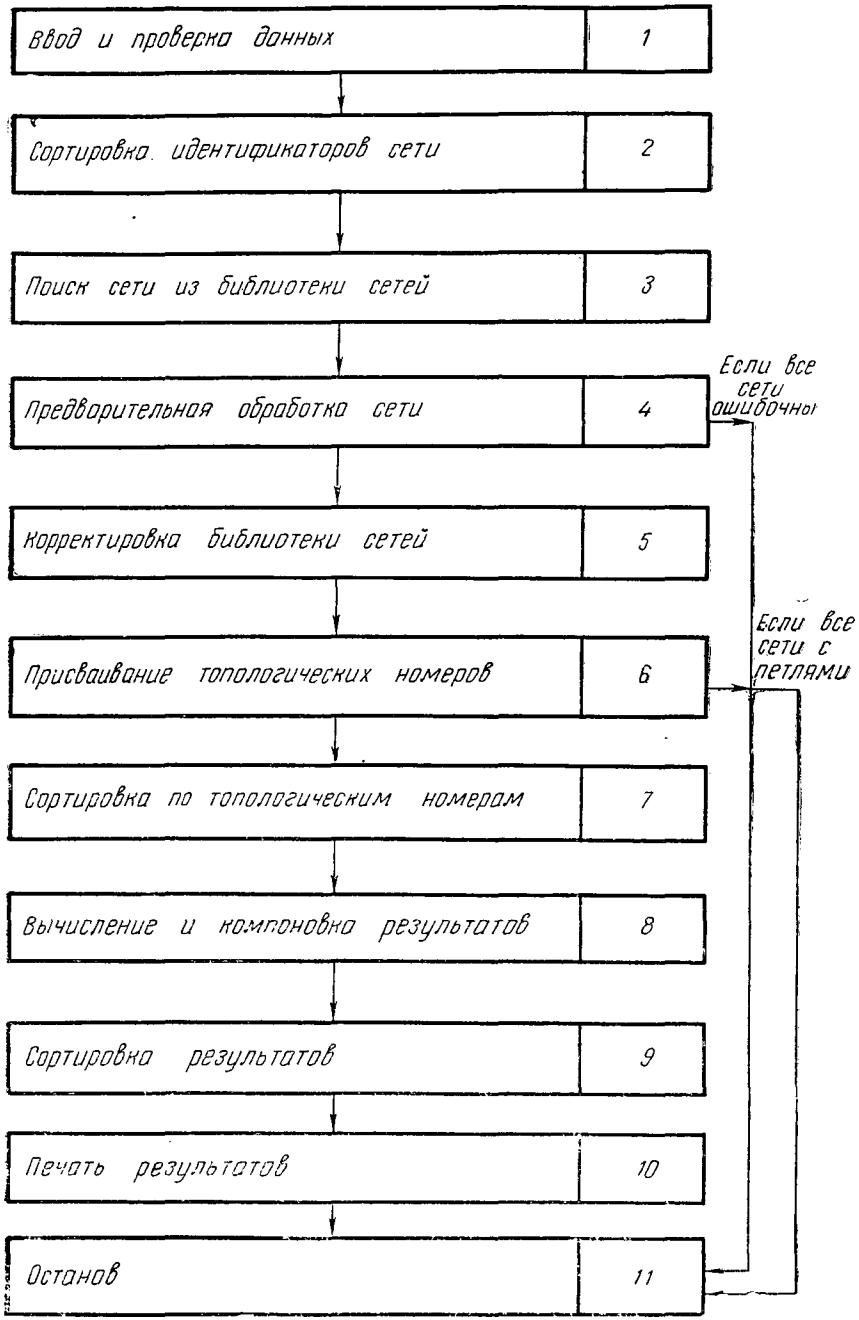


Рис. 20.1. Схема взаимосвязи программ в пакете ПЕРТ

2) сортировка идентификаторов сети;
3) поиск сети из библиотеки сетей;
4) предварительная обработка сети;
5) корректировка библиотеки сетей;
6) присваивание топологических номеров;
7) сортировка по топологическим номерам;
8) вычисления и компоновка результатов;
9) сортировка результатов;
10) печать результатов.

Схема взаимосвязи этих частей в процессе реализации пакета приведена на рис. 20.1.

Рассмотрим основные функции каждой программы пакета.

Программа ввода и проверки данных производит ввод данных с перфокарт или перфолент; проверку на приемлемость кода и формата последующими программами пакета; при необходимости делает сообщения об ошибках и выдает данные во вспомогательную память в форме, пригодной для сортировки.

Программа сортировки идентификаторов сети осуществляет сортировку идентификаторов сети в возрастающем порядке. Внутри каждой сети работы организуются в порядке «конец события — начало события», а события располагаются в порядке возрастания идентификаторов.

Программа поиска сети из библиотеки сетей производит поиск из библиотеки сетей сети с идентификатором, равным или больше идентификатора сети, подлежащего обработке. Процесс обработки сети зависит от типа прогона, определяемого соответствующим идентификатором прогона.

Первый прогон. Если идентификаторы равны, то входные данные обрабатываются, и печатается соответствующее сообщение. В противном случае входные данные проверяются на предмет правильности и непротиворечивости и записываются во вспомогательную память в форме, принятой для библиотеки сетей.

Повторный прогон. Если идентификаторы не равны, то входные данные отбрасываются, и печатается соответствующее сообщение. В противном случае входные данные считываются параллельно с данными, находящимися в библиотеке сетей. При этом входные данные проверяются на правильность и непротиворечивость с учетом данных, уже находящихся в библиотеке. Обновленный вариант сети переписывается во вспомогательную память. Если необходимо, изменения, внесенные в сеть, выводятся на печать.

Вычеркивающий прогон. Если идентификаторы не равны, то входные данные отбрасываются, и печатается соответствующее сообщение. В противном случае стираемая запись выносится в выходной массив.

Для каждой сети, подлежащей обработке, программа предварительной обработки сети обеспечивает присваивание псевдономеров каждому событию, проверку соответствия описания событий с

определениями в данных об операциях, сообщения об ошибках, запись сети во вспомогательную память.

Программа корректировки библиотеки сетей считывает предварительно обработанные сети (на четвертом этапе) и сети из библиотеки сетей с целью корректировки последней. Если имеются сети, которые не подвергаются изменениям, то они переписываются в выходную библиотеку сетей, а на печать выдаются подтверждающие сообщения. Новые и измененные сети переписываются из вспомогательной памяти в выходную библиотеку сетей и в массив сетей, подлежащих последующей обработке. Если сеть содержит ошибки, то она не переписывается в массив сетей для обработки. В этом случае в библиотеке сетей остается старый вариант сети. При необходимости обновленные сети выдаются на печать.

Каждому событию в рабочей сети программа присваивания топологических номеров присваивает топологические номера. Все события, предшествующие рассматриваемому, имеют меньшие номера, а все события, следующие за рассматриваемым, имеют большие номера, чем номер расматриваемого события. Программа сортировки по топологическим номерам осуществляет сортировку сокращенных записей работ в порядке топологических номеров.

Программа вычисления и компоновки результатов осуществляет все расчеты на сетях. С функциональной точки зрения ее можно разделить на две части: вычисления и компоновка выходных данных. В процессе вычислений сеть просматривается в прямом и обратном направлениях. Для каждого события определяются наиболее ранние и наиболее поздние даты выполнения, резервное время, критические предшественники, отклонения и другие требуемые параметры. Полученные результаты переписываются во вспомогательную память. Данные, полученные в результате вычислений, программой сортировки сортируются в порядке, указанном пользователем.

Программа печати результатов осуществляет вывод отсортированных результатов на печатающее устройство или во вспомогательную память.

Программы ПЕРТ требуют в качестве ввода не только данных, задающих сеть, но также значений ряда параметров. Примерами параметров могут служить тип прогона, идентификатор обрабатываемой сети, количество временных оценок, система отчетности и др.

Перечень параметров и их мнемонических обозначений зависит от конкретной реализации пакета.

ВОПРОСЫ К ГЛАВЕ

1. Что понимается под событием?
2. Что понимается под действием (операцией)?
3. Как строится сеть?
4. Какие оценки времени используются в сетях?
5. Как рассчитываются основные характеристики сети?
6. Назначение пакета ПЕРТ.

7. Виды корректировки сетей, находящихся в библиотеке.
8. Минимальный комплект устройств.
9. Состав пакета ПЕРТ.
10. Основные функции программ пакета.

Глава 21

ПАКЕТ ДЛЯ РЕШЕНИЯ ЗАДАЧ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ

21.1. Задача линейного программирования

Общая задача линейного программирования формулируется следующим образом.

Требуется найти вектор $(x_1, x_2, \ldots, x_j, \ldots, x_n)$, минимизирующий линейную функцию

$$L(x_1, x_2, \ldots, x_n) = c_1x_1 + c_2x_2 + \ldots + c_jx_j + \ldots + c_nx_n \quad (21.1)$$

при ограничениях, наложенных на переменные $x_1, \ldots, x_n$ вида:

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ &a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \qquad (21.2) \\ &\ldots\ldots\ldots\ldots\ldots\ldots \\ &a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n = b_i \\ &\ldots\ldots\ldots\ldots\ldots\ldots \\ &a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_n \qquad (21.3) \end{aligned}$$

и $x_j \geqslant 0$, $j = 1, 2, \ldots, n$,

где a_{ij}, b_i и c_j — заданные постоянные величины, а $m < n$.

В различных ситуациях удобно использовать разные виды записи общей задачи линейного программирования. Наиболее употребительными являются векторные формы записи. В векторной форме общая задача линейного программирования формулируется следующим образом.

Минимизировать CX при условии $AX = B$ и $X \geqslant 0$,

где $C = (c_1, c_2, \ldots, c_n)$ — вектор-строка, $X = (x_1, x_2, \ldots, x_n)$ — вектор-столбец, $A = \{a_{ij}\}$ — матрица, $B = (b_1, b_2, \ldots, b_m)$ — вектор-столбец и 0 — n-мерный нулевой вектор-столбец.

При анализе системы условий (21.2) и (21.3) могут представиться три случая:

1) условия (21.2) и (21.3) противоречивы, т. е. не существует набора чисел $x_1, x_2, \ldots, x_n$, удовлетворяющих всем условиям задачи;

2) условия (21.2) и (21.3) совместны, но область, определяемая ими, неограничена. Это означает, что существуют наборы чисел $x_1, x_2, \ldots, x_n$, удовлетворяющие системе (21.2) и (21.3) и содержащие отдельные переменные со сколько угодно большими значениями;

3) система условий (21.3) и (21.2) совместная и область, определяемая ею, ограничена.

Задача линейного программирования разрешима, если существует набор чисел $(x_1, x_2, \ldots, x_n)$ $(x_j < \infty)$, удовлетворяющий всем ограничениям (21.2) и (21.3) и обращающий линейную форму (21.2) в максимум или минимум.

Неразрешимость задачи может быть обусловлена либо противоречивостью условий задачи (случай 1), либо неограниченностью области определения линейной формы (случай 2).

В практике линейного программирования чаще других используется метод последовательного улучшения или симплексный метод. Основы этого метода были сформулированы Данцигом в 1949 г. При решении задачи этим методом можно выделить три существенных этапа:

1) вычисление опорного плана;
2) проверка выбранного опорного плана на оптимальность;
3) последовательное улучшение выбранного плана.

Задачи линейного программирования с дополнительными требованиями целочисленности решения называются задачами целочисленного программирования.

Задачи линейного программирования с дополнительными требованиями на пределы изменения исходных параметров задачи получили название задач параметрического программирования. Возможны и другие принципы классификации задач линейного программирования.

21.2. Краткая характеристика пакета

Пакет линейного программирования предназначен для решения задач линейного и целочисленного программирования. Пакет имеет сложную структуру, которая приведена на рис. 21.1.

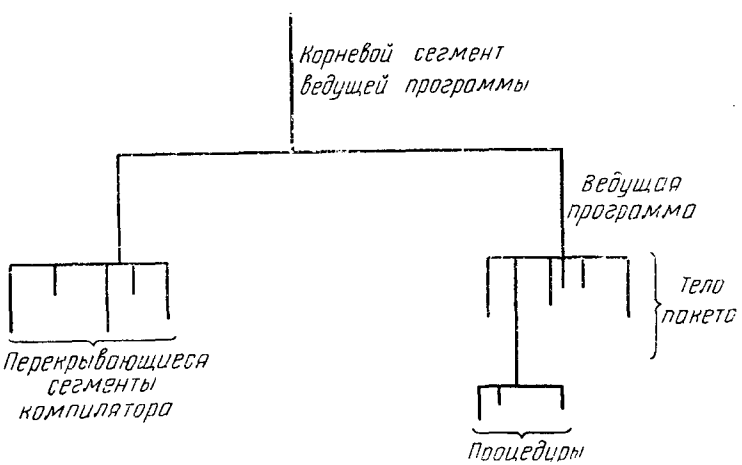


Рис. 21.1. Структура пакета ЛП

Во время выполнения пакета в основной памяти всегда находится корневой сегмент ведущей программы. В него входят управляющие таблицы пакета, области связей отдельных программ тела пакета, общие подпрограммы, буферы для хранения данных, а также программы распознавания операторов языка управления заданиями. Все другие сегменты загружаются в основную память СУПЕРВИЗОРом по требованию ведущей программы.

Определив на основании операторов языка управления заданиями, какую работу требуется выполнить (компилирование или исполнение), программа распознавания, находящаяся в корневом сегменте, подготавливает вызов требуемой программы. На входе системы может быть несколько типов заданий:

скомпилировать программу, написанную на входном языке пакета, и выполнить ее;

скомпилировать программу и сохранить для дальнейшего использования;

выполнить программу, которая была скомпилирована раньше.

Если поступило задание на компилирование, то программа распознавания вызывает корневой сегмент компилятора и передает ему управление. Компилятор производит трансляцию программы и записывает ее в библиотеку объектных модулей. По окончании работы управление передается программе распознавания. Если другие действия не требуются, то пакет закончил свою работу.

Если требуется выполнение полученной программы, то программа распознавания вызывает ведущую программу, которая будет определять дальнейшие действия. Если полученная программа находится в библиотеке объектных модулей, то необходимо вызвать РЕДАКТОР для редактирования объектного модуля и размещения его в библиотеке загрузочных модулей.

С этой целью ведущая программа выполняет команду обращения к СУПЕРВИЗОРу, который осуществляет прерывание. После возвращения управления ведущая программа на основании операторов входного языка пакета определяет, какую программу обработки следует вызвать в основную память для выполнения.

Программы обработки представляют собой тело пакета и находятся в библиотеке загрузочных модулей. Для вызова любой программы обработки ведущая программа также обращается к СУПЕРВИЗОРу. Кроме этого, при выполнении отдельных программ обработки возникают различного рода ошибки, которые требуют вмешательства оператора, или ситуации, требующие выполнения некоторых программ СУПЕРВИЗОРа.

Во входном языке пакета предусмотрено несколько операторов, определяющих прерывания. Все прерывания выполняет ведущая программа, а обрабатывает их СУПЕРВИЗОР. Для анализа результатов обработки прерываний СУПЕРВИЗОРом выделены специальные ячейки прерывания.

Программы обработки составляют тело пакета. Каждая из них имеет простую структуру или с запланированным перекрытием. По

окончании своей работы они возвращают управление ведущей программе, одновременно указывая ей, надо ли обратиться к СУПЕРВИЗОРу или нет. Они могут также выработать требование на вызов той или иной программы обработки.

Задача линейного программирования для решения ее с использованием пакета линейного программирования представляется в виде матрицы. Каждая строка и каждый столбец имеют описание. Оно состоит из трех частей: описания типа, описания названия и некоторого дополнительного описания.

Описание типа содержит информацию о характере обрабатываемой строки или столбца. Описание названия представляет собой некоторый идентификатор или некоторое имя. Каждая строка и каждый столбец должны иметь свое имя. Любая пара строк или столбцов не может иметь одно и то же имя. Дополнительное описание носит произвольный характер. Вводится ограничение на длину описания.

Каждой строке задачи линейного программирования ставится в соответствие логический вектор. В наиболее простом виде он состоит из столбца нулей за исключением элемента, находящегося на месте строки, которой он соответствует. Этот элемент может быть положительным, нулем, отрицательным или не иметь значения в зависимости от вида строки, которой он соответствует. В более сложных случаях логический вектор содержит информацию строки. Для регистрации результата отводится один столбец и одна строка. Описание должна иметь и вся задача в целом.

Для постановки задачи линейного программирования с использованием пакета ЛП необходимо наличие двух видов файлов информации. Один из них должен содержать данные задачи, второй — операторы входного языка. Файл данных задачи состоит из описания задачи, описания строк, описания столбцов и матричных элементов.

Файл операторов входного языка представляет собой программу, написанную пользователем на входном языке пакета для решения конкретной задачи линейного программирования.

Минимальная конфигурация машины для использования пакета линейного программирования может быть задана в следующем виде:

одно считывающее устройство с карт,

одно устройство широкоформатной печати,

два диска,

одно устройство магнитной ленты,

один перфоратор карт,

одна консоль оператора.

Если пакет работает под управлением дисковой операционной системы, то требования к памяти можно сформулировать таким образом:

дисковая операционная система	20 К
память под программы ЛП	30 К
память под данные, включая буферную память 5 К . . .	13 К
Всего . . .	63 К

Эти требования предъявляются при решении задач линейного программирования малых размеров, т. е. число строк матрицы $\leqslant 200$.

Для решения больших задач с количеством строк порядка 1000 к памяти предъявляются следующие требования:

дисковая операционная система	20 К
память под программы ЛП	30 К
память под данные, включая буферную память 88 К . . .	193 К
Всего . . .	243 К

21.3. Входной язык пакета

Входной язык служит для написания управляющих программ. Это язык высокого уровня, с помощью которого пользователь легко и быстро может определить стратегию решения задач.

Программа, написанная на входном языке, называется управляющей программой пакета и предназначена для выбора стратегии решения задачи пользователя. Операторы языка пакета могут определить вычисления, запросы на прерывания, вызывать процедуры.

Во входном языке используются следующие группы знаков: числа ($0 \div 9$), алфавит ($A \div z$), разделители ($\lrcorner$, ., (,), =, +, —, /, $*$), специальный символ γ. В качестве элементов языка используются константы, переменные, операторы.

Константы и переменные могут иметь тип: целый, вещественный, текстовой, адресный. Текстовая константа заключается в строчные кавычки (апострофы), которые при этом не являются частью константы. Накладываются определенные ограничения и на длину используемых в языке констант. Переменные могут быть простыми и с индексами.

По участию в работе программы все операторы можно разделить на исполняемые и неисполняемые.

Исполняемые операторы переводятся в последовательность команд рабочей программы, а неисполняемые операторы являются носителями информации для компилятора и для исполнительных команд. В рабочей программе этим операторам не соответствуют никакие команды. К числу неисполняемых операторов относятся, например, операторы начала и конца управляющей программы, операторы описания данных. К исполняемым операторам относятся арифметические операторы, операторы передачи управления, операторы процедуры.

Рассмотрим кратко общую характеристику и назначение операторов управляющего языка.

Арифметические операторы типа ВЫПОЛНИТЬ могут быть простой и сложной формы. Простой оператор ВЫПОЛНИТЬ присваивает переменной значение некоторой другой переменной. Сложный оператор ВЫПОЛНИТЬ вычисляет значение арифметического выражения и присваивает это значение переменной. Все переменные в поле операнда должны быть одного типа. Если операнды являются текстовыми или адресными константами, то арифметический оператор должен быть простым.

К операторам управления можно отнести операторы, определяющие начало и конец управляющей программы, и операторы передачи управления.

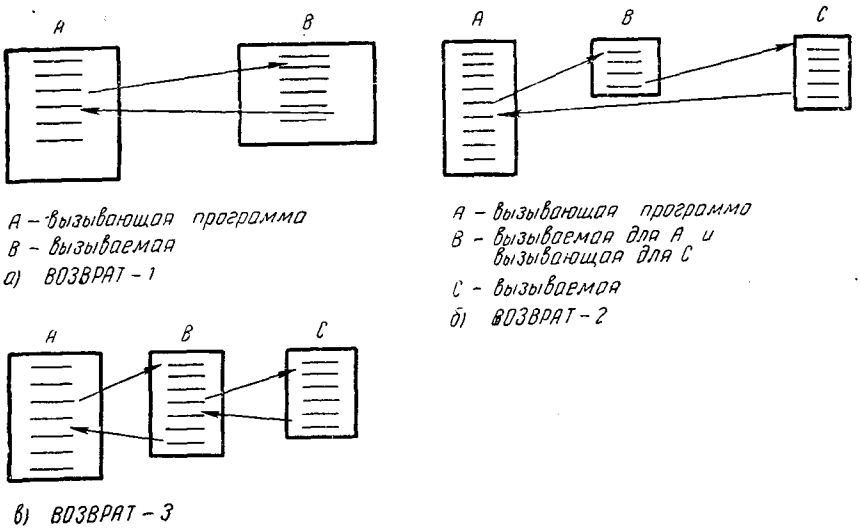


Рис. 21.2. Иллюстрация действий оператора ВОЗВРАТ:
а — ВОЗВРАТ-1: А — вызывающая программа, В — вызываемая программа; *б* — ВОЗВРАТ-2: А — вызывающая програма, В — вызываемая программа для А и вызывающая для С, С — вызываемая программа; *в* — ВОЗВРАТ-3

Начальный оператор управляющей программы будем обозначать идентификатором НАЧАЛО, а заключительный оператор — КОНЕЦ. Как и в обычных алгоритмических языках, в языке пакета могут использоваться операторы безусловной передачи управления например ПЕРЕЙТИ, оператор условной передачи управления ЕСЛИ и пустой оператор ПУСТО.

Кроме того, могут использоваться специальные операторы передачи управления стандартным подпрограммам. Эти операторы в отличие от безусловных и условных операторов передачи управления могут предусматривать выполнение ряда операций перед тем, как передать управление стандартной подпрограмме.

Наибольший интерес среди операторов передачи управления представляют те, которые обеспечивают возврат к вызывающим программам разных уровней от вызываемых программ. Назовем эти операторы ВОЗВРАТ. Можно выделить четыре разновидности таких операторов: ВОЗВРАТ-1, ВОЗВРАТ-2, ВОЗВРАТ-3, ВОЗВРАТ-4. Схема, иллюстрирующая действие операторов ВОЗВРАТ, приведена на рис. 21.2.

Оператор ВОЗВРАТ-1 передает управление от вызываемой программы к вызывающей по окончании работы вызываемой программы. Оператор ВОЗВРАТ-2 передает управление в первую вызываемую программу, минуя непосредственно вызывающую программу, как это показано на рис. 21.2б. Оператор ВОЗВРАТ-3 обеспечивает передачу управления первой вызывающей программе, последовательно возвращая управление на всех уровнях вызова, как показано на рис. 21.2в. Оператор ВОЗВРАТ-4 вызывает прекращение выполнения вызываемой и вызывающей программ и передает управление СУПЕРВИЗОРу.

Необходимость иметь различные типы оператора ВОЗВРАТ диктуется двумя соображениями. С одной стороны, операционная система представляет проблемным программам аппарат передачи управления между различными программами или частями программ стандартным образом посредством макрокоманд RETURN (ВОЗВРАТ). Поэтому нет смысла дублировать его в каждой проблемной программе.

С другой стороны, в отдельных программах пакета возникают ситуации, которые должны обрабатываться по-разному и которые требуют иногда нарушения обычной последовательности выполнения программ. Чтобы не создавать для таких ситуаций аппарат передачи управления, можно воспользоваться тем, который предоставляется операционной системой, но предварительно надо произвести настройку на правильную обработку этих ситуаций СУПЕРВИЗОРом и правильную передачу управления после обработки таких ситуаций. Реализуется это тем, что для каждой ситуации существуют свои ячейки прерывания, служащие для связи с СУПЕРВИЗОРом, а выполняемый оператор ВОЗВРАТ формирует команду обращения к СУПЕРВИЗОРу с нужными параметрами.

При выполнении программ пакета ЛП возникают следующие ситуации, требующие вмешательства СУПЕРВИЗОРа и вызывающие прерывания: ситуации ошибок; нормальные ситуации при решении задач; ситуации, подготовленные пользователем для работы в определенных временных режимах или с определенной частотой итерации, или для действий, производимых с пульта оператора; ситуации, связанные с выбором стратегии решения задачи. Будем называть их алгоритмическими запросами.

Среди ошибок, вызывающих запросы на прерывание, различаются:

главная ошибка — ошибка в постановке задания. Не установ-

лены соответствующим образом ячейки обратной связи или недоступны затребованные файлы;

текущая ошибка ввода-вывода — ошибка при вводе или выводе данных;

вероятная ошибка ввода — ошибка, возникшая в одной из вводных процедур;

возможная ошибка ввода — ошибка, замеченная во время одной из вводных процедур, но точно не установленная;

ошибка в вычислениях.

Запросы, связанные с ситуацией анормального решения, вызываются следующими ошибками:

имеет место вырожденность; в этом случае, восстановление прежнего состояния может произойти автоматически или пользователь может предпринять собственные действия;

нет допустимого решения — задача снимается с решения;

решение неограниченное — решение принимает оператор;

отсутствие максимальных значений параметров в параметрических алгоритмах;

непредусмотренный максимум, устанавливаемый в параметрических алгоритмах.

Для осуществления запросов по требованию пользователя необходимо установить соответствующие ячейки условия или соответствующий запрос на прерывание должен быть послан с пульта управления. Наиболее употребительны следующие запросы:

запросы, определяемые интервалами времени с целью сохранения и восстановления данных при неисправностях машины;

запросы по максимуму времени с целью соблюдения порядка окончания программы;

запросы по частоте итераций;

запросы печати решения во время выполнения параметрических алгоритмов.

Алгоритмические запросы — это множество запросов, которые предусматриваются для выбора стратегии решения конкретной задачи, описанной пользователем. Например, некоторые процедуры могут потребовать выполнения других процедур перед своим продолжением в зависимости от указанных действий на входном языке пакета.

Каждый запрос должен иметь соответствующий идентификатор, представляющий собой адрес переменной, содержащей метку первого оператора соответствующей программы прерывания. Каждая программа прерывания должна заканчиваться соответствующим оператором возврата.

Операторы описаний используются для описания данных и для размещения комментариев. Все данные, перечисленные в операторе описания, должны иметь один и тот же тип.

Операторы процедуры служат для вызова процедур. В отличие от обычного алгоритмического языка в языке пакета ЛП могут использоваться процедуры на уровне языка пакета и на уровне языка

управления заданиями. Для отличия первого уровня процедур от второго процедуры в языке управления должны иметь специальные операторы начала и конца процедуры. Заголовок и тело процедуры имеют такую же структуру, как и в алгоритмических языках.

В процедурах, используемых в пакете линейного программирования в качестве параметров процедур, могут использоваться список выбора строк и список выбора столбцов. Если присутствуют оба списка, то список строк должен предшествовать списку столбцов.

Оператор процедуры является основным аппаратом входного языка пакета, обеспечивающим связь с телом пакета. Тело пакета представляет собой набор процедур, обеспечивающих подготовку и ввод данных, вычисление отдельных результатов, вывод результатов, восстановление информации и другие функции.

Процедуры тела пакета могут быть записаны на различных алгоритмических языках. Наиболее часто для записи алгоритмов процедур используются ассемблерные языки, а также языки ФОРТРАН, АЛГОЛ, КОБОЛ.

21.4. Описание процедур пакета

В пакете ЛП можно выделить ряд процедур ввода данных, обеспечивающих ввод, проверку, перевод в соответствующие системы счисления исходных данных, корректировку уже существующих файлов, формирование столбцов и строк матрицы.

В качестве алгоритмических процедур можно выделить процедуры анализа постановки задачи, построения базиса, обращения матрицы, процедуры оптимизации, проверки решений. Можно выделить процедуры, выполняющие компоновку и копирование различных файлов, вывод результатов и др.

Описание каждой процедуры должно начинаться указанием идентификатора процедуры и списка формальных параметров. Идентификатор должен соответствовать наименованию процедуры в теле пакета.

Если для выполнения процедуры необходимы определенные условия, например наличие некоторых данных, программ или проверка некоторых условий, то в области связи должны быть выделены специальные поля. Для каждого поля области связи должно быть указано его содержание и условия использования.

Если для выполнения процедуры необходимо предварительное использование других процедур, то в процедуру должна быть включена проверка выполнения предварительных процедур.

В описании процедур указываются основные действия, выполняемые процедурой, возможный перечень ошибок с описанием их причин и условия прерывания выполнения процедур. Указываются также дополнительные сообщения, которые могут быть получены от системы в ходе реализации каждой процедуры.

Прежде чем рассматривать примеры описаний отдельных процедур, приведем перечень файлов, используемых пакетами ЛП в

процессе реализации задач. Каждому из файлов поставим в соответствие идентификатор с тем, чтобы использовать его при описании процедур.

При постановке задачи линейного программирования исходными являются: файл данных ФД и файл управляющей программы ФУП. На основании данных файла ФД и данных, сформированных процедурами, могут формироваться проблемные файлы ФП, которые размещаются в библиотеках данных системы ЛП.

В процессе решения задачи дополнительно могут образовываться и использоваться матричный файл ФМ, файл обратной матрицы ФМО, файл печатающего устройства ФПУ, файл перфорирующего устройства ФПР и рабочие файлы, которые могут иметь различные обозначения в зависимости от размещения. Условимся все рабочие файлы обозначать ФР. Скомпилированная управляющая программа также образует несколько специальных файлов. Они могут иметь несколько обозначений в зависимости от средств размещения (диски, ленты). Условимся обозначать их идентификатором ФКУП.

Каждый из перечисленных файлов имеет свою структуру. Рассмотрим в качестве примера структуру проблемного файла, который включает таблицу содержания проблемного файла, зону непосредственного доступа и зону последовательного доступа.

Таблица содержания проблемного файла обеспечивает индексирование данных в зоне непосредственного доступа и содержит относительные адреса внешней памяти в зоне последовательного доступа. Таблица создается вместе с созданием проблемного файла, и размеры ее не могут быть изменены. Каждому элементу данных из зоны непосредственного доступа в таблице соответствует запись, содержащая идентификатор, описание и относительный адрес начала данных на носителе.

Зона непосредственного доступа, следующая за таблицей, — содержание проблемного файла — содержит элементы данных, индексированных в таблице. Элементы данных в зоне непосредственного доступа определяются путем последовательного просмотра таблицы.

Зона последовательного доступа располагается после последнего элемента зоны непосредственного доступа. Она представляет собой область расширения проблемного файла. Элемент данных в этой зоне определяется путем нахождения начала области расширения по адресу в таблице содержания проблемного файла и последующего поиска соответствующего идентификатора.

Рассмотрим примеры описания некоторых процедур, осуществляющих ввод, вычисления и вывод.

Процедура ВВОД предназначена для создания проблемного файла ФП из файла данных ФД. Идентификатор этой процедуры может быть записан в следующем виде:

ВВОД (A = наименование данных B, C, D, E, F, M).

Параметры в процедуре имеют следующее назначение. Присут-

ствие параметра (А-наименование данных) означает, что наименование данных будет именем требуемого множества данных. Если параметр А не присутствует, то в качестве имени требуемого множества данных берется содержимое специального поля ДАННЫЕ. Присутствие в процедуре параметра В означает, что проверку на дублирование наименования столбцов проводить не нужно. Присутствие параметра С означает отмену сообщения ИМЯ НЕ ОПРЕДЕЛЕНО.

Наличие параметра Д исключает проверку наименований на ведущие нули и нулевые значения. Наличие параметра Е обеспечивает печать числа элементов по строке и по столбцу для каждого вектора. На этот случай в описании процедуры должен быть соответствующий формат.

Наличие параметра F вызывает запись в проблемном файле наименования строк с границами, функционала и правой части вместе с их последовательными номерами.

Параметр М представляет собой список выбора строк и столбцов. Он определяет векторы, которые должны быть введены для решения задачи.

Для выполнения процедуры ВВОД в области связи должны быть выделены соответствующие поля, содержащие наименования задачи, проблемного файла, количество карт, содержащихся в одном блоке магнитной ленты (при использовании магнитных лент для хранения файла данных), указатель наличия создания проблемного файла и наименование данных в том случае, если оно отсутствует в качестве параметра процедуры.

При выполнении процедуры ВВОД могут возникать прерывания, вызванные следующими ошибками: главная ошибка, текущая ошибка ввода-вывода, вероятная ошибка ввода, возможная ошибка ввода. Прерывание может иметь место и в случае окончания данных.

Сообщения об ошибках выдаются на устройство печати. Каждое сообщение об ошибке выдается с ошибочной картой и относительным последовательным номером этой карты в файле ФД. В каждом сообщении указаны причины ошибки, если таковые имеются, и выполняемые действия.

При исполнении процедуры ВВОД могут выдаваться различные сообщения об ошибках. Рассмотрим примеры таких сообщений.

ПАМЯТЬ НЕДОСТАТОЧНА

Причиной такого сообщения может служить недостаточность места в памяти для буферов. В этом случае устанавливается причина прерывания ГЛАВНАЯ ОШИБКА, после чего выполнение процедуры прекращается. В таблицу содержания проблемного файла записей не производится.

ОШИБОЧНЫЙ КОНЕЦ ФАЙЛА

Причиной такого сообщения служит обнаружение конца файла данных раньше фактического окончания данных. В этом случае усганавливается причина прерывания ГЛАВНАЯ ОШИБКА. Вы-

полнение процедуры прекращается. В таблицу содержания проблемного файла производятся соответствующие записи.

КОРРЕКТИРОВАТЬ ФД

Причиной сообщения может быть исправление в файле данных или нулевое содержание поля, в котором должно содержаться имя данных, при отсутствии наименования в качестве параметра процедуры. При появлении такого сообщения устанавливается причина прерывания ГЛАВНАЯ ОШИБКА, выполнение процедуры останавливается, ввода данных не происходит.

ОШИБОЧНЫЙ ТИП ВЕКТОРА

Причиной может служит недопустимый тип вектора. Устанавливается причина прерывания ВЕРОЯТНАЯ ОШИБКА ВВОДА.

ВЕКТОР БЕЗ НАИМЕНОВАНИЯ

Причиной такого сообщения может служить пустое поле наименования вектора. Устанавливается прерывание ВЕРОЯТНАЯ ОШИБКА ВВОДА, и все карты вектора, поле наименования которых пусто, игнорируются.

Кроме сообщений об ошибках и действиях, связанных с ними, могут выдаваться различного рода пояснительные сообщения. Примерами таких сообщений могут служить следующие:

успешно записан проблемный файл,

ГЛАВНАЯ ОШИБКА имела место до завершения ввода.

Сообщение о состоянии задачи будет выдано прежде чем управление будет передано ведущей программе пакета.

При выполнении процедуры ВВОД используются четыре разновидности файлов: файл данных, проблемный файл, рабочий файл, файл на печатающем устройстве.

Процедура ОБРАЩЕНИЕ обеспечивает вычисление обратной матрицы текущего базиса и записывает полученную матрицу в файл обратной матрицы ФМО. Обращение к этой процедуре имеет вид:

ОБРАЩЕНИЕ

Таким образом, данная процедура параметров не имеет. Перед процедурой обращения должна быть выполнена процедура МАТРИЦА, которая создает матричный файл МФ. Для выполения процедуры в области связи должны быть выделены поля для размещения:

минимального абсолютного значения главного элемента. Если значение главного элемента вектора меньше содержимого данного поля, вектор исключается из базиса;

вспомогательного числа для получения значения главного элемента;

числа, указывающего точность представления нулевых элементов. Все элементы, имеющие меньшее значение, чем число в данном поле, помещаются в файл обратной матрицы как нулевые.

В процедуре могут возникнуть две причины прерываний:

текущая ошибка ввода-вывода,

имеет место вырожденность.

В качестве сообщений об ошибках могут использоваться сообщения вида:

НЕДОСТАТОК ПАМЯТИ

Причиной такого сообщения может быть недостаток места, отведенного под рабочие файлы или под файл обратной матрицы. В этом случае устанавливается причина прерывания ТЕКУЩАЯ ОШИБКА ВВОДА-ВЫВОДА, и выполнение процедуры прекращается.

БАЗИС ВЫРОЖДЕННЫЙ

Причиной такого сообщения служит вырожденность базиса. По этому сообщению устанавливается соответствующая причина прерывания, и вырожденный вектор удаляется из базиса. В данной процедуре используются матричный файл, файл обратной матрицы, рабочие файлы и файл на печатающем устройстве.

Процедура ОПТИМИЗАЦИЯ предназначена для оптимизации решения задачи. Оптимизация выполняется в два этапа:

I этап — нахождение допустимого решения,

II этап — оптимизация полученного решения.

В процедуре используется итерационный метод оптимизации. Обращение к процедуре имеет вид:

ОПТИМИЗАЦИЯ

Никаких параметров процедура не имеет. Перед ее выполнением должны быть выполнены процедуры МАТРИЦА и ОБРАЩЕНИЕ. Процедура ОПТИМИЗАЦИЯ прежде всего проверяет выполнение процедуры МАТРИЦА. Если она по каким-либо причинам не выполнена, устанавливается соответствующая причина прерывания, например МАТРИЦЫ НЕТ, и выполнение процедуры ОПТИМИЗАЦИЯ прекращается. Если процедура МАТРИЦА выполнена, проверяется наличие файла обратной матрицы. Если такого файла нет, устанавливается прерывание, указывающее на невыполнение процедуры обращения матрицы.

Если проверка в обоих случаях прошла успешно, текущее решение приемлемо для процедуры ОПТИМИЗАЦИЯ.

Для выполнения процедуры ОПТИМИЗАЦИЯ в области связи должны быть выделены поля для указания вида оптимизации (минимизации или максимизации решения), количества итераций, максимума времени решения, частоты вызова процедуры ОБРАЩЕНИЕ, наименования функционала, наименования правой части, наименования измененной правой части и других параметров.

В процессе выполнения процедуры могут выдаваться сообщения об ошибках следующего характера:

отсутствие в матричном файле какой-либо информации,

решение неограниченное,

допустимое решение отсутствует,

истекло максимальное время решения,

выполнено установленное число итераций и другие сообщения.

Так как процедура ОПТИМИЗАЦИЯ использует итерационный метод оптимизации, то каждый раз число итераций предсказать

трудно. Поэтому при выполнении процедуры предусматривается специальное поле, содержащее число итераций, и время решения.

В процедуре используются следующие файлы: матричный файл, проблемный файл, рабочие файлы и файл на печатающее устройство.

Процедура РЕШЕНИЕ обеспечивает вывод текущего решения на печатающее устройство. Обращение к ней имеет вид:

РЕШЕНИЕ (*X*, *Y*)

Наличие параметра *X* обеспечивает распечатку базисных строк и столбцов. Наличие параметра *Y* указывает на то, что должны быть распечатаны выбранные строки и столбцы. Если параметры *X* и *Y* не указаны, должны быть распечатаны все решения.

Перед выполнением процедуры РЕШЕНИЕ должна быть выполнена проверка, что задача решена и что текущий базис находится в файле обратной матрицы. Если ошибок нет, то процедура РЕШЕНИЕ считывает векторы в соответствии с установленными параметрами *X* и *Y*. Когда все векторы считаны в память, производится перевод номеров в наименования векторов с соответствующими описаниями. После этого решение выводится на печатающее устройство.

Выводимый файл должен содержать наименование задачи, номера страниц, наименование каждого столбца и соответствующую информацию. Для вывода результатов на печать должен быть задан формат печати.

В процессе выполнения данной процедуры могут возникать прерывания по причинам: главная ошибка, текущая ошибка ввода-вывода, вероятная ошибка ввода или ошибка перечня выбора. В качестве сообщений об ошибках могут быть использованы сообщения следующего содержания:

НЕТ ФУНКЦИОНАЛА

Причиной такого сообщения может служить отсутствие в матричном файле строки функционала. В этом случае устанавливается причина прерывания ГЛАВНАЯ ОШИБКА, и выполнение процедуры прекращается.

НЕТ ПРАВОЙ ЧАСТИ

Причиной такого сообщения может быть отсутствие в секции файла правой части. В случае такого сообщения устанавливается прерывание ГЛАВНАЯ ОШИБКА, и выполнение процедуры останавливается.

НЕТ ГРАНИЦЫ

Такое сообщение может появиться в том случае, если не может быть найдена верхняя (нижняя) граница пары границ. При появлении такого сообщения устанавливается причина прерывания ВЕРОЯТНАЯ ОШИБКА ВВОДА и указывается действие, которое необходимо выполнить, например пара границ игнорируется.

В качестве пояснительных сообщений могут выдаваться сообщения вида: ошибок не обнаружено, решение выдано; во время выполнения процедуры РЕШЕНИЕ имела место ошибка; во время

выполнения возникли прерывания (дается перечень причин прерываний); прерываний не было.

При выполнении данной процедуры используются три типа файлов: матричный файл, файл обратной матрицы и файл на печатающее устройство.

Процедура ПЕРФОРАЦИЯ обеспечивает вывод базиса на перфокарты в одинаковой форме. Обращение к процедуре имеет вид:

ПЕРФОРАЦИЯ (А-«наименование»)

Параметр процедуры А определяет наименование множества выводимых данных. В области связи для процедуры ПЕРФОРАЦИЯ должны быть отведены два поля: для указания наименования задач и наименования проблемного файла.

Перед выполнением процедуры ПЕРФОРАЦИЯ должно быть проверено выполнение процедуры МАТРИЦА для задачи, наименование которой указано в соответствующем поле области связей. При несоблюдении проверяемых условий устанавливается причина прерывания ГЛАВНАЯ ОШИБКА, и выполнение процедуры прекращается.

При отсутствии ошибок все наименования строк из матричного файла считываются в память. Считывается в память также базис, состоящий из множества последовательных номеров векторов. Номера векторов преобразуются в наименования векторов.

На карте перфорируется наименование каждого базисного вектора, его тип, наименование вектора строки, который должен быть заменен базисным вектором. Базис, выведенный на перфорирующее устройство процедурой ПЕРФОРАЦИЯ, может быть считан как файл данных.

В процедуре ПЕРФОРАЦИЯ могут возникать прерывания двух видов: главная ошибка, текущая ошибка ввода-вывода. В качестве сообщений об ошибках могут иметь место сообщения вида:

НЕТ ДАННЫХ

Такое сообщение может появиться в случае, когда процедура МАТРИЦА не вызвана перед процедурой ПЕРФОРАЦИЯ.

ОШИБКА ВВОДА

Сообщение появляется при обнаружении ошибок при вводе или выводе информации. В случае появления этого сообщения устанавливается причина прерывания ТЕКУЩАЯ ОШИБКА ВВОДА-ВЫВОДА, и выполнение процедуры останавливается.

В качестве пояснительных сообщений можно использовать сообщения вида: процедура ПЕРФОРАЦИЯ окончена, прерываний не установлено; ошибок при выполнении процедуры не обнаружено; перечень установленных причин прерываний.

Примеры описаний приведенных процедур в конкретном исполнении могут быть выполнены и в другой форме. Это особенно относится к перечню сообщений об ошибках, к причинам возникновения прерываний и к пояснительным сообщениям. Данное описание имело целью демонстрацию общих принципов описания и его организационной структуры.

21.5. Подготовка исходных файлов

Для решения задачи линейного программирования с использованием пакета ЛП требуется подготовить файл данных и файл управляющей программы с помощью форматных карт. Для ввода этих файлов в систему и выполнения управляющей программы подготовленные файлы оформляются с помощью операторов языка управления заданиями, образуя задание на выполнение задачи линейного программирования. Рассмотрим структуру и назначение форматных карт каждого файла.

Файл данных может содержать карты следующих четырех типов:

1. Индикаторные карты: а) карта-указатель процедуры; б) карта окончания данных; в) карта-указатель секции; г) подындикаторная карта.
2. Карты-комментарии.
3. Титульные карты.
4. Карты данных.

Индикаторные карты должны содержать тип указателя и имя. Они используются для вызова соответствующих процедур ввода данных, для указания отдельных составных частей данных, имеющих самостоятельное значение (секций), для указания конца множества данных. Эти карты всегда предшествуют картам с данными.

Весь файл данных состоит из записей разного типа. Одной записи соответствует одна карта. Записи одного типа объединяются и оформляются с помощью индикаторных карт. Первой стоит карта с указателем процедуры, которая предназначена для ввода и обработки данного типа карт. Последней является карта окончания данных или карта указателя процедуры для следующей последовательности карт данных.

Среди записей одного типа выделяются отдельные секции, содержащие карты данных строк, карты данных столбцов или карты правой части. В этом случае каждая секция должна иметь соответствующий указатель секции: СТРОКА, СТОЛБЕЦ, ПРАВАЯ ЧАСТЬ.

Подындикаторные карты служат для различных модификаций данных. Модификации могут подлежать как проблемные файлы, так и карты данных. В проблемном файле можно модифицировать существующие вектора, вводить новые вектора, изменять порядок расположения векторов, стирать их содержимое. На картах данных могут быть изменены: тип вектора, описание вектора, масштабный коэффициент, значение элемента вектора. Титульные карты могут быть использованы для размещения заголовка данных. Карты-комментарий содержат различные комментарии. Карты данных предназначены для ввода информационных данных.

Каждому типу карты соответствует свой формат. Форматы карт составляются в процессе разработки пакета. Соблюдение форматов карт при пользовании пакетом обязательно.

Рассмотрим примеры содержания форматных карт для каждого из перечисленных типов.

Карта-указатель процедуры содержит наименование процедуры ввода, наименование данных и порядковый номер карты, разделенные пробелами. Карта окончания данных содержит КОНЕЦ ДАННЫХ, пробелы и порядковый номер карты. Карта-указатель секции содержит наименование секции данных, пробелы и порядковый номер карты.

Подындикаторная карта содержит наименование подындикатора, наименование вектора или пробелы, порядковый номер карты, разделенные пробелами. Карта-комментарий содержит специальный признак, комментарий и порядковый номер карты. Карты-комментарии могут быть двух типов: карты, содержащие комментарий, который игнорируется процедурами обработки, и карты, содержащие комментарий для формирования файла на печатающем устройстве. Специальный признак служит для отличия этих карт.

Титульная карта содержит ТИТУЛ, пробелы, конкретное содержание титула и порядковый номер карты. Карта данных содержит:

(поле 1) тип вектора,
(поле 2) наименование вектора,
(поле 3) наименование элемента,
(поле 4) значение элемента,
(поле 5) наименование элемента,
(поле 6) значение элемента,
порядковый номер карты.

Вектор определяется одной или более последовательными картами данных. Тип вектора берется из поля 1 первой карты вектора. На последующих картах вектора это поле может быть заполнено пробелами, в противном случае оно должно быть идентично содержимому поля 1 первой карты. Наименование вектора в поле 2 должно быть повторено во всех последующих картах. Отсутствие наименования в поле 2 воспринимается как начало следующего вектора. На карте данных может быть помещено не более двух элементов вектора.

Ввод данных начинается при обнаружении индикаторной карты указателя процедуры и заканчивается при появлении карты окончания данных или другой карты указателя процедуры. Следующей картой за картой указателя процедуры может быть титульная карта. Если процедура ввода допускает разделение данных на секции, то данные определенной секции следуют непосредственно за индикатором секции. Порядок расположения секций должен быть строго определен. Например, первой всегда располагается секция СТРОКА, за ней секция СТОЛБЕЦ, после чего следует секция ПРАВАЯ ЧАСТЬ.

Любые титульные карты после первой карты ТИТУЛ игнорируются. Карты-комментарии могут размещаться в любом месте.

Порядок расположения форматных карт в файле данных рассмотрим на следующем примере:

ВВОД	Указатель процедуры
ТИТУЛ	Заголовок
СТРОКА	Указатель секции строк
———	
———	
———	
———	Карты данных строк
СТОЛБЕЦ	Указатель секции столбцов
———	
———	
———	
———	Карты данных столбцов
ПРАВАЯ ЧАСТЬ	Указатель секции правой части
———	
———	
———	
———	Карты данных правой части
КОНЕЦ ДАННЫХ	Указатель окончания данных

Формат карт и порядок расположения форматных карт в файле данных может изменяться в зависимости от используемых процедур ввода и их специфики в конкретном пакете прикладных программ.

Управляющая программа представляет собой совокупность операторов входного языка пакета. Управляющая программа начинается с оператора НАЧАЛО и заканчивается оператором КОНЕЦ. Кроме операторов входного языка пакета, управляющая программа может содержать различные комментарии. Таким образом, файл управляющей программы может содержать форматные карты двух типов: операторные карты и карты-комментарий.

Операторные карты предназначены для записи операторов языка. Каждый оператор программы размещается на отдельной карте. Форматы карт для всех операторов одинаковы. Если по синтаксису языка представляются операторы с длиной, превышающей одну карту, должна быть предусмотрена возможность продлить его на другую карту. В этом случае должен быть задан формат карты-продолжения.

Рассмотрим примеры форматных карт файла управляющей программы.

Операторная карта содержит через пробелы метку, наименование оператора, операнды, признак продолжения и порядковый номер карты.

Карта-продолжения содержит пробелы, операнды (продолжение с предыдущей карты), признак продолжения и порядковый номер карты. В пакете должно быть указано максимальное число карт, которое может занимать один оператор.

Карта-комментарий содержит специальный признак, комментарий, пробел и порядковый номер карты.

Форматная карта-комментарий может иметь одинаковый формат для файла данных и для файла управляющей программы. В нашем примере эти карты имеют различные форматы.

21.6. Общая схема работы пакета

Выполнение задачи линейного программирования с использованием пакета ЛП начинается с компиляции управляющей программы, составленной пользователем. Компиляция управляющей программы осуществляется компилятором, который вызывается в основную память ведущей программой пакета.

Компилятор читает управляющую программу из соответствующего файла, размещенного на перфокартах или магнитной ленте, выполняет проверку синтаксиса, компиляцию и выводит скомпилированную управляющую программу. Эта программа может быть использована для последующих решений.

В зависимости от имеющегося состава оборудования скомпилированная программа может быть записана в библиотеке объектных модулей на дисках или на магнитных лентах в виде блоков фиксированной длины. Такие блоки называют операторными блоками. Каждый операторный блок содержит целое число операторов.

Кроме операторных блоков, скомпилированная управляющая программа должна содержать в самом начале блок описаний и в конце — блок словаря. Блок описаний должен содержать начальные значения всех переменных управляющей программы. Блок словаря должен содержать адрес каждого операторного блока вместе с номером его оператора.

После компиляции при условии отсутствия ошибок может начаться выполнение программы. Оно производится также под управлением ведущей программы пакета, корневой сегмент которой постоянно находится в памяти. Перед выполнением управляющей программы ведущая программа считывает в основную память блок описаний и блок словаря. Эти блоки остаются в памяти на все время решения.

Затем ведущая программа считывает в память первый операторный блок и выполняет его. В процессе решения задачи в основной памяти может находиться в каждый момент времени только один операторный блок. Перед выполнением нового оператора ведущая программа должна проверить наличие оператора в текущем операторном блоке. Если его там нет, из словаря определяется адрес требуемого блока, который считывается в основную память.

Когда выполняется оператор вызова процедуры, загрузка нужной процедуры производится в область перекрытия на место старой процедуры. Все процедуры, которые вызываются при решении, имеют доступ к области связи и к общим файлам на внешних устройствах.

Когда выполнение процедуры заканчивается, она передает управление ведущей программе, чтобы определить, были ли установ-

лены причины прерывания. Если ведущая программа обнаруживает причины прерывания, она выполняет соответствующую программу прерывания, которая подготавливает информацию для СУПЕРВИЗОРа, для правильной обработки прерывания и возврата управления в нужное место.

Если процедура заканчивается и причина прерываний не установлена, то выполняется следующий оператор управляющей программы.

При выполнении ведущей программы в случае обнаружения ошибок могут выдаваться специальные сообщения о характере ошибок. Примерами таких сообщений могут служить следующие:

указанной процедуры нет в данной версии пакета ЛП;

значение метки слишком велико;

Сделана попытка деления, а пара переменных в управляющей программе неопределенна.

ВОПРОСЫ К ГЛАВЕ

1. Назначение пакета ЛП и его краткая характеристика.
2. Минимальная конфигурация машины.
3. Характеристика входного языка пакета ЛП.
4. Состав процедур пакета ЛП.
5. Подготовка исходных файлов в пакете ЛП.

Глава 22

ПАКЕТ МАТЕМАТИЧЕСКОЙ ОБРАБОТКИ НАБЛЮДЕНИЙ

22.1. Математическая постановка задачи

Статистический анализ результатов экспериментов (наблюдений) позволяет решить ряд практически важных задач. Во-первых, обеспечивается объективная оценка изучаемого процесса (процессов) по результатам наблюдений. Во-вторых, выявляются статистические закономерности изучаемого процесса с точки зрения получения количественной и качественной оценки. В-третьих, на основе полученных оценок обеспечивается построение моделей прогнозирования и принятия решений.

Одним из важнейших требований к конструкции пакета программ математической обработки наблюдений является обеспечение работы пользователя в режиме непосредственного диалога с машиной.

Такой режим позволяет исследователю по ходу обработки данных выбирать путь продолжения обработки на основе некоторой совокупности промежуточных результатов.

Характеристики одномерных рядов распределения

Основные свойства распределения измеряются соответствующими численными характеристиками, обеспечивающими выполнение оценки качества по каждому из одномерных рядов. Наиболее распространенными числовыми характеристиками рядов распределения являются средняя арифметическая, дисперсия, среднее квадратическое отклонение, коэффициент вариации, показатели асимметрии и эксцесса.

Выборочное среднее, являющееся приближенной оценкой теоретического среднего (математического ожидания) генеральной совокупности, определяется по формуле

$$\overline{X}_j = \frac{1}{n} \sum_{i=1}^{n} X_{ij} \text{ при } j = 1, 2, \ldots k. \tag{22.1}$$

Средняя арифметическая как показатель среднего уровня, вокруг которого колеблются значения членов вариационного ряда, является важной, но недостаточной характеристикой распределения. Это уточнение может быть получено при знании среднего квадратического или стандартного отклонения, определяемого для каждого столбца по формуле

$$S_j = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (X_{ij} - \overline{X}_j)^2}. \tag{22.2}$$

Величина

$$S_j^2 = \frac{1}{n} \sum_{i=1}^{n} (X_{ij} - \overline{X}_j)^2 \tag{22.3}$$

называется выборочной дисперсией.

Величины S_j и S_j^2 дают хотя и состоятельные, но смещенные оценки для теоретических значений среднего квадратического отклонения σ и дисперсии σ^2. В связи с этим при небольших выборках для вычисления применяют несколько измененные выражения, дающие несмещенные оценки для σ и σ^2, т. е. значения $\overline{S}_j$ и $\overline{S}_j^2$ вычисляются по формулам:

$$\overline{S}_j^2 = \frac{1}{n-1} \sum_{i=1}^{n} (X_{ij} - \overline{X}_j)^2 \tag{22.4}$$

и

$$\overline{S}_j = \sqrt{\overline{S}_j^2}. \tag{22.5}$$

Оценки средней арифметической и дисперсии для случайной выборки позволяют при условии нормального распределения утверждать, что в границах, определяемых как $(\overline{X}-S, \overline{X}+S)$, содержится приблизительно 68% всех наблюдений, в границах $(\overline{X}-2S, \overline{X}+2S)$ размещаются уже 95% всех наблюдений и почти все наблюдения (99,7%) укладываются в трехсигмовый интервал.

Ошибка среднего значения вычисляется по формуле

$$S_{\overline{X}_j} = \frac{\sigma_j}{\sqrt{n-1}} = \frac{S_i}{\sqrt{n}}. \qquad (22.6)$$

Величина ошибки среднего значения используется для определения радиуса и границ доверительного интервала среднего значения

$$\Delta\overline{X}_j = t\,(n-1)\,S_{\overline{X}_j}; \qquad (22.7)$$

$$X_j^{(1)} = \overline{X}_j - \Delta\overline{X}_j; \qquad (22.8)$$

$$X_j^{(2)} = \overline{X}_j + \Delta\overline{X}_j, \qquad (22.9)$$

где $\Delta\overline{X}_j$ — радиус доверительного интервала среднего значения j-го признака;

$t_{(n-1)}$ — значение t-критерия Стьюдента для $n-1$ степени свободы в выбранном уровне значимости;

X_j^1, X_j^2 — левый и правый концы доверительного интервала.

В случае, если имеет место значительное отклонение распределения от нормального, эти характеристики не дают качественной оценки. В качестве меры отклонения распределения от нормального используются показатели асимметрии (скошенности) и эксцесса (островершинности).

Эти характеристики получаются расчетным путем на основании следующих зависимостей:

$$k_{1j} = \frac{\sqrt{n\,(n-1)}}{n-2} \cdot \frac{M_{3i}}{M_{2j}^{3/2}} \qquad (22.10)$$

и

$$k_{2j} = \frac{n-1}{(n-2)\,(n-3)}\left[(n+1)\left(\frac{M_{4j}}{M_{2j}^2} - 3\right) + 6\right]. \qquad (22.11)$$

Здесь M_{2j}, M_{3j}, M_{4j} — центральные моменты соответственно второго, третьего и четвертого порядка.

Приблизительно гипотезу о принадлежности распределения к нормальному можно проверить с помощью показателей асимметрии и эксцесса. Принято считать, что если $k_1 \leqslant 3\sigma_{k_1}$ и $k_2 \leqslant 5\sigma_{k_2}$, то наблюдаемое распределение можно отнести к нормальному.

Средние квадратические отклонения для характеристик k_1 и k_2 соответственно равны:

$$S_{k_1} = \sqrt{\frac{6n\,(n-1)}{(n-2)\,(n+1)\,(n+3)}} \qquad (22.12)$$

и

$$S_{k_2} = \sqrt{\frac{24n\,(n-1)^2}{(n-3)\,(n-2)\,(n+3)\,(n+5)}}. \qquad (22.13)$$

Из других числовых характеристик часто используется коэффициент вариации, характеризующей степень рассеяния индивидуальных значений членов ряда по отношению к среднему уровню. Коэффициент вариации выражается в процентах и вычисляется по формуле

$$v_j = \frac{\overline{S_j}}{\overline{X_j}} \cdot 100\% \qquad (22.14)$$

Ошибка коэффициента вариации определяется по формуле

$$\sigma_{v_j} = \frac{v_j}{\sqrt{n}} = \sqrt{0.5 + v_j^2}. \qquad (22.15)$$

В связи с тем что нормальное распределение играет особую роль при изучении характеристик одномерных рядов распределения, необходимо включить в алгоритм обработки данных проверку соответствия выборочного распределения нормальной форме. Кроме анализа показателей асимметрии и эксцесса, может быть выполнена проверка нормальности распределения на основании критериев Пирсона, Колмогорова и др.

Одним из распространенных критериев является критерий Пирсона (χ^2), вычисляемый по формуле

$$\chi_j^2 = \Sigma \frac{(m_j - m_j')^2}{m_j'}, \qquad (22.16)$$

где m_j — эмпирические частоты j-го признака;

m_j' — теоретические частоты j-го признака.

Критерий χ^2 применяется для оценки анормальных дискретных распределений. На практике удобно использовать нормальное распределение и для характеристики дискретных распределений, что во многих случаях является оправданным.

Для непрерывных распределений критерий χ^2 может дать неудовлетворительные результаты из-за неудачного выбора числа и длины интервалов, искажающих истинный характер распределения. Поэтому его применение носит ограниченный характер.

В противоположность критерию χ^2, применение которого связано с группировкой исходных данных по интервалам, критерий Колмогорова основывается на несгруппированных значениях величин. Критерий Колмогорова рекомендуется к применению только для проверки непрерывных распределений.

Вычислительная процедура проверки нормальности распределения должна обеспечивать получение характеристик применительно к трем перечисленным выше критериям с тем, чтобы исследователь сам принял окончательное решение о выборе критерия.

На основе полученных характеристик частотных распределителей одномерных рядов можно сделать заключение о качестве материала для последующей обработки.

Математические методы, при помощи которых исследуются и обобщаются взаимосвязи варьирующих признаков, называются регрессионным и корреляционным анализом. Методы анализа регрессии и корреляции позволяют выявить количественные закономерности и измерить тесноту связей между изучаемыми признаками.

Наиболее простым типом уравнений связи являются линейные уравнения, которые графически изображаются прямой линией. Линейная функция выражает арифметическую пропорциональность: увеличению (уменьшению) аргумента или фактора на одну единицу всегда соответствует одинаковый прирост (убывание) функции или результатного признака.

На практике встречается большое количество взаимосвязей, которые заключаются в пропорциональном изменении среднего значения зависимого признака с изменением факторного. Если в прямоугольной системе координат нанести точки, ординаты и абсциссы которых будут соответственно значениями факторного и результатного признаков, то полученная совокупность точек представляет поле корреляции.

В корреляционном поле можно провести прямую линию так, чтобы все точки находились по возможности ближе к этой линии, т. е. сумма отклонений была бы минимальной. Уравнение этой линии (линии регрессии) в общем виде можно записать так:

$$y = b_0 + b_1 x, \tag{22.17}$$

где b_0 — длина отрезка, отсекаемого на оси ординат;

b_1 — тангенс угла наклона линии регрессии к оси абсцисс.

Это уравнение, выражающее общую закономерность взаимосвязей между признаками x и y, называется *уравнением регрессии*. Для определения коэффициентов уравнения регрессии используется метод наименьших квадратов. В основу этого метода положено требование, чтобы сумма квадратов отклонений фактических величин от расчетных по уравнению была бы минимальной. Математически это требование может быть записано так:

$$Q = \sum_{i=1}^{n} (y_i - \tilde{y}_i)^2 \to \min. \tag{22.18}$$

Для определения констант уравнения прямой $\tilde{y}_i = b_0 + b_1 x_i$, отвечающей требованию метода наименьших квадратов, заменим y_i его значением:

$$Q = \sum_{i=1}^{n} (y_i - b_0 - b_1 x_i)^2.$$

Полученное выражение будет принимать минимальное значение при

$$\frac{\partial Q}{\partial b_0} = 0 \text{ и } \frac{\partial Q}{\partial b_1} = 0, \text{ т. е.}$$

$$\left.\begin{aligned} \frac{\partial Q}{\partial b_0} &= -2\sum_{i=1}^{n}(y_i - b_0 - b_1 x_i) \\ \frac{\partial Q}{\partial b_1} &= -2\sum_{i=1}^{n}(y_i - b_0 - b_1 x_i)\, x_i \end{aligned}\right\}$$

или

$$\left.\begin{aligned} \sum_{i=1}^{n} y_i &= nb_0 + b_1 \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} y_i x_i &= b_0 \sum_{i=1}^{n} x_i + b_1 \sum_{i=1}^{n} x_i^2 \end{aligned}\right\}. \qquad (22.19)$$

Система уравнений (22.19), в которой b_0 и b_1 неизвестны, называется системой нормальных уравнений для выбранного уравнения регрессии (22.17). После решения системы уравнений (22.19) относительно b_0 и b_1 уравнение регрессии (22.17) становится вполне определенным.

В качестве меры колеблемости фактических величин вокруг расчетных (ошибки оценки) принимается остаточное среднее квадратическое или стандартное отклонение фактических величин y_i от расчетных $\tilde{y}_i$. Стандартную ошибку оценки по уравнению регрессии, вычисляемую по данным генеральной совокупности, обозначают через

$$\sigma_{yx} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2}{n}}. \qquad (22.20)$$

Вычисление средней квадратической ошибки выборки производится по формуле

$$S_{yx} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2}{n-2}}, \qquad (22.21$$

где $(n-2)$ — число степеней свободы.

Потеря двух степеней свободы вызвана использованием двух условий при определении уравнения регрессии. Для больших выборок $n \approx n-2$ потерю степеней свободы можно не учитывать. Тогда

$$S_{yx} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2}{n}}. \qquad (22.22)$$

Остаточное среднее квадратическое отклонение используется для построения доверительных границ для оценок по уравнению регрессии. Для этих целей выбирается вероятность P_1 в качестве гарантии того, что оцениваемая величина y не будет ниже оценки $\tilde{y}$ более чем на $t_{P_1} S_{yx}$ и выбирается вероятность P_2, которая гарантирует верхнюю границу для $\tilde{y}$. Тогда верхняя доверительная граница запишется $(\tilde{y}+t_{P_2} S_{yx})$, а нижняя — $(\tilde{y}-t_{P_1} S_{yx})$. Величины t_{P_1} и t_{P_2} находятся по выбранным значениям P_1 и P_2 по таблицам t-критерия Стьюдента, если считать распределение вероятностей ошибок $y-\tilde{y}$ близким к распределению Стьюдента.

Тогда доверительные границы можно записать в виде следующего неравенства:

$$b_0 + b_1 x_i - t_{P_1} S_{yx} \leqslant y \leqslant b_0 + b_1 x_1 + t_{P_2} S_{yx}.$$

В случае $P_1 = P_2$ доверительные границы выражаются симметричными линиями относительно линии регрессии. Установленные доверительные границы отражают неточность предсказания результативного признака, обусловленную необъясненной вариацией, так как связь не функциональная, а корреляционная.

Доказано, что общая дисперсия состоит из двух частей.

$$\sigma_y^2 = \sigma_{\tilde{y}}^2 + \sigma_{yx}^2. \qquad (22.23)$$

Она объясняется регрессией и определяется:

$$\sigma_{\tilde{y}}^2 = \frac{\sum_{i=1}^{n} (\tilde{y}_i - \tilde{y})^2}{n}. \qquad (22.24)$$

Разложение суммы квадратов отклонений на составные части дает возможность разработать методы оценки, известные под названием дисперсионного анализа. Для выборочных данных равенство (22.23) становится приблизительным:

$$S_y^2 \approx \sigma_y^2 + S_{yx}^2. \qquad (22.25)$$

Остаточное среднее квадратическое отклонение выражается в тех же единицах, что и первоначальные значения y_i. Поэтому для различных уравнений стандартные ошибки оценки по уравнениям не сопоставимы. Для сопоставления используются отвлеченные показатели взаимосвязей.

Одним из показателей, характеризующим долю объясненной дисперсии во всей дисперсии, служит отношение

$$R^2 = \frac{\sigma_{\tilde{y}}^2}{S_y^2}, \qquad (22.26)$$

называемое коэффициентом детерминации.

В качестве относительной меры того, насколько изменение зави-

симой переменной обусловлено влиянием изменения аргумента, рассматривается коэффициент корреляции:

$$R = \sqrt{\frac{\sigma^2_{\tilde{y}}}{S^2_y}}.$$

Из уравнения (22.25) следует:

$$\sigma^2_{\tilde{y}} = S_y{}^2 - S^2_{yx}.$$

Следовательно, коэффициент корреляции равен:

$$R = \sqrt{1 - \frac{S^2_{yx}}{S^2_y}}. \qquad (22.27)$$

В общем случае коэффициент корреляции может принимать значения в диапазоне

$$-1 \leqslant R \leqslant 1.$$

Высокое по модулю значение коэффициента корреляции соответствует более тесной взаимосвязи зависимой переменной и аргумента.

Стандартная ошибка коэффициента регресии в выборках, выполненных в нормально распределенной или близкой к ней совокупности, оценивается по формуле

$$S_b = \frac{S_{yx}}{S_x\sqrt{n}}. \qquad (22.28)$$

Доверительные границы для уровней вероятности P_1 и P_2 можно записать так:

$$b_1 - S_b t_{P_1} \leqslant \beta \leqslant b_1 + S_b t_{P_2},$$

где β — коэффициент регрессии генеральной совокупности.

Доверительные границы индивидуального предсказания зависимой переменной можно определить для уровней вероятности P_1 и P_2 так:

$$\tilde{y}_{x_i} - t_{P_1} S_{\bar{y}x_i} \leqslant \tilde{y}_{x_i} \leqslant \tilde{y}_{x_i} + t_{P_2} S_{\bar{y}x_i}.$$

Стандартная ошибка коэффициента корреляции при больших выборках оценивается по формуле

$$\sigma_r = \frac{1 - \rho^2}{\sqrt{n-1}},$$

где ρ — коэффициент корреляции в генеральной совокупности. В связи с тем, что значение ρ неизвестно, его заменяют коэффициентом корреляции, вычисленным по выборочной совокупности:

$$S_R = \frac{1-R^2}{\sqrt{n-1}}. \qquad (22.29)$$

При определении существенности коэффициента корреляции выдвигается гипотеза, по которой коэффициент корреляции генеральной совокупности ρ принимается равным нулю и определяется, совместимо ли это предположение с выборочным коэффициентом корреляции R. Если $\rho=0$, то

$$\sigma_r = \frac{1}{\sqrt{n-1}}.$$

Для проверки нулевой гипотезы рекомендуется зависимость

$$\mu = \frac{|R|}{\sigma_r}. \qquad (22.30)$$

Если $\mu \geqslant 2.58$, то гипотеза отвергается, корреляция признается существенной. Если $\mu < 2.58$, то нулевая гипотеза принимается, и взаимосвязь признается ненадежной.

Метод наименьших квадратов — основной метод анализа регрессии и корреляции. Одним из вспомогательных методов анализа регрессии и корреляции является метод моментов. В этом методе применяются другие зависимости, которые дают результаты, соответствующие методу наименьших квадратов, но более удобные при составлении программы на ЭВМ. При использовании метода моментов расчеты ведутся без учета потерь степеней свободы, поэтому оценки S_x, S_y, S_{yx} получаются смещенными.

Если выборка большая, то $n \approx n-1$, если же маленькая, то S_x, S_y, S_{xy} уточняются при помощи специальных поправочных коэффициентов. При условии, что начало отсчета взято в точке с координатами $\bar{x}$ и $\bar{y}$, нормальные уравнения (22.19) примут вид:

$$\left.\begin{aligned} \sum_{i=1}^{n}(y_i-\bar{y}) &= nb_0 + b_1\sum_{i=1}^{n}(x_i-\bar{x}) \\ \sum_{i=1}^{n}(y_i-\bar{y})(x_i-\bar{x}) &= b_0\sum_{i=1}^{n}(x_i-\bar{x}) + b_1\sum_{i=1}^{n}(x_i-\bar{x})^2 \end{aligned}\right\}. \qquad (22.31)$$

Но так как

$$\sum_{i=1}^{n}(x_i-\bar{x}) = 0 \text{ и } \sum_{i=1}^{n}(y_i-\bar{y}) = 0,$$

то из первого уравнения системы получим $b_0=0$.

Второе уравнение имеет вид:

$$\sum_{i=1}^{n}(y_i-\bar{y})(x_i-\bar{x}) = b_1\sum_{i=1}^{n}(x_i-\bar{x})^2.$$

Коэффициент регрессии в этом случае определяется:

$$b_1 = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}. \qquad (22.32)$$

Вычисление коэффициента корреляции с учетом несмещенных оценок $S^2_{y.x}$ и σ_y^2 можно выполнить по формулам:

$$S^2_{y.x} = \frac{\sum_{i=1}^{n} y_i^2 - b_0 \sum_{i=1}^{n} y_i - b_1 \sum_{i=1}^{n} x_i y_i}{n} \qquad (22.33)$$

и

$$\sigma_y^2 = \frac{\sum_{i=1}^{n} y_i^2}{n} - \bar{y}_i^2. \qquad (22.34)$$

Значение коэффициента детерминации имеет вид:

$$R^2 = \frac{b_0 \sum_{i=1}^{n} y_i + b_1 \sum_{i=1}^{n} x_i y_i - n\bar{y}^2}{\sum_{i=1}^{n} y_i^2 - n\bar{y}^2}. \qquad (22.35)$$

Если начало координат расположено в точке $(\bar{x}, \bar{y})$, то $\bar{y}^2 = 0$ и формула (22.35) имеет вид:

$$R^2 = \frac{b_1 \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (y_i - \bar{y})^2}. \qquad (22.36)$$

Значение b_1 можно подставить из формулы (22.32). Тогда

$$R = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2 \sum_{i=1}^{n} (y_i - \bar{y})^2}}. \qquad (22.37)$$

Из определения дисперсии следует, что

$$\sum_{i=1}^{n} (y_i - \bar{y})^2 = n\sigma_y^2.$$

и

$$\sum_{i=1}^{n} (x_i - \bar{x})^2 = n\sigma_x^2.$$

Следовательно,

$$R = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})}{n\sigma_x\sigma_y}. \qquad (22.38)$$

Средняя величина произведений $(y_i - \bar{y})(x_i - \bar{x})$ определяется для $i = 1, 2, \ldots, n$ и называется ковариацией, или первым смешанным моментом:

$$C_{yx} = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})}{n}. \qquad (22.39)$$

Отсюда

$$R = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})}{n\sigma_y\sigma_x}. \qquad (22.40)$$

Эта зависимость является центральной формулой при вычислении показателей корреляции по способу моментов. Формулу ковариации (22.39) можно преобразовать к виду, удобному для вычислений:

$$C_{yx} = \frac{\sum_{i=1}^{n} y_i x_i}{n} - \bar{y}\bar{x}. \qquad (22.41)$$

Из зависимости (22.36) можно получить значение

$$b_1 = \frac{R^2 \sum_{i=1}^{n} (y_i - \bar{y})^2}{\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})}$$

или, используя зависимость (22.40), получить

$$b_1 = R\frac{\sigma_y}{\sigma_x}. \qquad (22.42)$$

Полученная зависимость также является одной из основных формул метода моментов.

Зависимость для определения b_0 получается из первого уравнения системы нормальных уравнений (22.19):

$$b_0 = \frac{\sum_{i=1}^{n} y_i}{n} - b_1 \frac{\sum_{i=1}^{n} x_i}{n}. \qquad (22.43)$$

Для вычисления стандартной ошибки оценки по уравнению регрессии преобразуется зависимость (22.27) к виду:

$$S_{yx} = \sigma_y \sqrt{1 - R^2}. \qquad (22.44)$$

Стандартные ошибки и доверительные границы коэффициента регрессии, корреляции, среднего и индивидуального предсказания находятся по тем же формулам, что и при методе наименьших квадратов.

Если выборка небольшая, а необходимо иметь точные оценки доверительных границ, стандартные отклонения и ошибки уточняются с учетом потерь степеней свободы. При этом используются следующие зависимости:

$$S_y^2 = \sigma_y^2 \frac{n}{n-1};$$

$$S_x^2 = \sigma_x^2 \frac{n}{n-2};$$

$$\overline{S}_{yx}^2 = S_{yx}^2 \frac{n}{n-2};$$

$$R = \sqrt{1 - \frac{S_{yx}^2}{\sigma_y^2} \cdot \frac{n-1}{n-2}}.$$

При статистическом анализе регрессии и корреляции приходится выяснять взаимосвязи не двух, а трех и более признаков. Применение метода линейной регрессии и корреляции и изучение взаимосвязей всех признаков попарно в одном случае может оказаться недостаточным, так как при исследовании взаимосвязи между зависимой переменной и одним фактором не позволяет выявить ее в чистом виде.

Взаимосвязь трех и более признаков часто выражается при помощи линейного уравнения регрессии. Линейное уравнение множественной регрессии отличается от уравнения простой регрессии дополнительными коэффициентами, число которых определяется количеством независимых переменных. В общем случае при k независимых переменных уравнение регрессии имеет вид:

$$\tilde{y} = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_k x_k. \qquad (22.45)$$

Система нормальных уравнений для множественной регрессии будет иметь вид:

$$\left.\begin{aligned}
\sum_{i=1}^{n} y_i &= b_0 n + b_1 \sum_{i=1}^{n} x_{i1} + b_2 \sum_{i=1}^{n} x_{i2} + \ldots + b_k \sum_{i=1}^{n} x_{ik} \\
\sum_{i=1}^{n} y_i x_{i1} &= b_0 \sum_{i=1}^{n} x_{i1} + b_1 \sum_{i=1}^{n} x_{i1}^2 + b_2 \sum_{i=1}^{n} x_{i1} x_{i2} + \ldots + b_k \sum_{i=1}^{n} x_{i1} x_{ik} \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\sum_{i=1}^{n} y_i x_{ik} &= b_0 \sum_{i=1}^{n} x_{ik} + b_1 \sum_{i=1}^{n} x_{i1} x_{ik} + \ldots + b_k \sum_{i=1}^{n} x_{ik}^2
\end{aligned}\right\} \qquad (22.46)$$

В результате решения системы нормальных уравнений относительно b_0, b_1, $b_2, \ldots, b_k$ получаются значения коэффициентов урав-

нения множественной регрессии. Следовательно, для нахождения коэффициентов уравнения необходимо иметь суммы данных всех признаков, суммы их квадратов и суммы всех парных произведений.

Систему нормальных уравнений для вычисления коэффициентов множественной регрессии можно составить и на основе расчетных величин — сумм отклонений от средних арифметических, дисперсий и ковариаций, коэффициентов парной корреляции. Так, при вычислениях по отклонениям от средних арифметических система нормальных уравнений имеет вид:

$$\left.\begin{aligned}
&\sum_{i=1}^{n}(y_i-\bar{y}) = b_0 n + b_1\sum_{i=1}^{n}(x_{i1}-\bar{x}_1) + b_2\sum_{i=1}^{n}(x_{i2}-\bar{x}_2)+\\
&\qquad + \ldots + b_k\sum_{i=1}^{n}(x_{ik}-\bar{x}_k)\\
&\sum_{i=1}^{n}(y_i-\bar{y})(x_{i1}-\bar{x}_1) = b_0\sum_{i=1}^{n}(x_{i1}-\bar{x}_1) + b_1\sum_{i=1}^{n}(x_{i1}-\bar{x}_1)^2 +\\
&+ b_2\sum_{i=1}^{n}(x_{i1}-\bar{x}_1)(x_{i2}-\bar{x}_2) + \ldots + b_k\sum_{i=1}^{n}(x_{i1}-\bar{x}_1)(x_{ik}-\bar{x}_k)\\
&\sum_{i=1}^{n}(y_1-\bar{y})(x_{ik}-\bar{x}_k) = b_0\sum_{i=1}^{n}(x_{ik}-\bar{x}_k) + b_1\sum_{i=1}^{n}(x_{i1}-\\
&-\bar{x}_1)(x_{ik}-\bar{x}_k) + b_2\sum_{i=1}^{n}(x_{i2}-\bar{x}_2)(x_{ik}-\bar{x}_k) + \ldots +\\
&\qquad + b_k\sum_{i=1}^{n}(x_{ik}-\bar{x}_k)^2
\end{aligned}\right\}. \qquad (22.47)$$

После соответствующих упрощений система (22.47) может быть решена методом обратных матриц. Уравнение множественной линейной регрессии не выражает функциональной зависимости, при которой каждой комбинации факторов соответствовала бы строго определенная величина результатного признака. Значение этого признака, определяемого по уравнению множественной регрессии, при любой комбинации факторов $x_1, x_2, \ldots, x_k$ является лишь наиболее вероятной величиной ожидаемого результата. Следовательно, необходимо вычислить ошибку оценки по уравнению и меру тесноты связи.

Стандартную ошибку оценки по уравнению множественной регрессии можно вычислить следующим образом:

$$S_{y,\,1,\,2,\,\ldots,\,k} = \sqrt{\frac{\sum_{i=1}^{n}(y_i-\tilde{y}_i)^2}{n-k-1}}. \qquad (22.48)$$

В качестве относительной меры, насколько изменение зависимой переменной обусловлено влиянием изменения аргументов, принято рассматривать коэффициент множественной корреляции:

$$R_{y,\,1,\,2,\,\ldots\,,\,k}=\sqrt{1-\frac{S^2_{y,\,1,\,2,\,\ldots\,,\,k}}{S^2_y}}. \qquad (22.49)$$

Доверительные границы оценки множественной регрессии и индивидуального предсказания оцениваются аналогично случаю простой линейной регрессии, только t_P берется для $n-k-1$ степеней свободы.

При изучении влияния нескольких факторов на результатный признак большое значение имеют частная регрессия и корреляция. В этом случае выясняется взаимосвязь, которая существует между одним из факторов и результатом при условии, что остальные факторы остаются неизменными. Для получения уравнения частной регрессии в уравнении множественной регрессии вместо неизменных факторов подставляют их средние значения, тогда

$$\tilde{y}_{j,\,1,\,2,\,\ldots\,,\,k}=b_0+b_1\bar{x}_1+b_2\bar{x}_2+\ldots+b_k\bar{x}_k. \qquad (22.50)$$

Коэффициенты частной регрессии совпадают с соответствующими коэффициентами множественной регрессии.

Коэффициенты частной корреляции показывают тесноту связи между результатным признаком и каким-либо факторным признаком при закреплении на неизменном уровне значений других факторов. Расчетная зависимость для получения коэффициентов частной корреляции имеет вид:

$$R^2_{yj,1,2,\,\ldots,\,(j-1),\,(j+1),\,\ldots,\,k}=\frac{S^2_{y,\,1,\,2,\,\ldots,\,(j-1),\,(j+1),\,\ldots,\,k}-S^2_{y,\,1,\,2,\,\ldots\,,\,k}}{S^2_{y,\,1,\,2,\,\ldots\,,\,(j-1),\,(j+1),\,\ldots\,,\,k}}. \qquad (22.51)$$

Знаменатель дроби представляет собой остаточную дисперсию результатного признака после того, как уже было учтено влияние всех факторов, кроме j-го. Второй член числителя — это остаточная дисперсия при учете влияния признака x_j. Числитель показывает снижение остаточной дисперсии за счет добавления к факторам и фактора x_j.

22.2. Краткая характеристика пакета

Пакет программ математической обработки наблюдений должен выполнять следующие функции:

1) ввод, редактирование и размещение исходных данных во внешней памяти;

2) вывод любого заданного числа строк исходной матрицы данных;

3) исключение резко выделяющихся наблюдений на основе анализа одномерных рядов распределения;

4) построение одномерных частотных распределений с автоматическим выбором числа интервалов;

5) расчет числовых характеристик одномерных рядов распреде-

лений: средних арифметических, дисперсий, показателей асимметрии и эксцесса, коэффициентов вариации и т. д.;

6) вычисление соответствующих величин для проверки нормальности распределений по различным критериям;

7) преобразования над столбцами матрицы исходных данных с целью получения по методу наименьших квадратов оценок для коэффициентов регрессии практически любого вида уравнения, линейного относительно оцениваемых параметров;

8) повторение пунктов 2—6 для преобразованной матрицы;

9) вычисление коэффициентов регрессии для нормированных и натуральных значений переменных;

10) получение матриц коэффициентов парной и частной корреляции;

11) вычисление характеристик, необходимых для проведения статистического анализа уравнения регрессии (множественный коэффициент корреляции, частные и множественные коэффициенты детерминации, остаточную дисперсию, значения t-критерия для коэффициентов регрессии, коэффициенты корреляции между коэффициентами регрессии и другие характеристики);

12) последовательный отсев несущественных факторов по наименьшему значению t-отношения величины коэффициента регрессии к его среднеквадратической ошибке или отсев любого из факторов по желанию исследователя, т. е. реализация схемы многошагового регрессионного анализа;

13) построение доверительных интервалов для расчетных значений функции по уравнению, признанному окончательным в процессе отсева несущественных факторов;

14) вычисление показателей, характеризующих качество модели: относительная и средневзвешенная ошибки аппроксимации, среднеквадратическое отклонение;

15) вывод результатов вычислений в заданном объеме в целях документирования.

Пакет включает ведущую программу, три модуля, программы из библиотеки стандартных функций и процедуры.

Ведущая программа работает на основании управляющих операторов входного языка, среди которых выделим следующие.

Наличие карты оператор ПРЕОБРАЗОВАНИЕ указывает на необходимость произвести преобразование матрицы. Отсутствие карты указывает на то, что преобразований производить не надо.

Оператор МОДЕЛЬ задает вид модели посредством параметров ПРЯМАЯ, СТЕПЕННАЯ, КОЛИЧЕСТВО. Отсутствие этой карты расценивается ведущей программой как ошибка.

Оператор ХАРАКТЕРИСТИКА задает необходимость вычислений числовых характеристик одномерных рядов распределения.

Оператор ИСКЛЮЧИТЬ указывает на необходимость исключения резковыделяющихся наблюдений.

Оператор ПАРНКОР определяет необходимость вычислений парной корреляции.

Оператор КОЭФ задает необходимость вычисления коэффициентов уравнения регрессии, оценки, множественной корреляции.

Оператор ЧАСТНКОР задает вычисление частной корреляции.

Оператор ОТСЕВ указывает на необходимость произвести отсев факторов.

Оператор ОТКЛОНЕНИЕ обозначает необходимость вычисления значения функции отклонения и коэффициентов аппроксимации.

Существуют и другие операторы, определяющие вычисления регрессионного и корреляционного анализа.

Исходная информация для работы пакета программ может вводиться с устройства чтения карт, с устройства ввода с перфолент или находится в библиотеке вводных данных. Тип устройства ввода задается в предложении *DD* языка управления заданиями.

Ведущая программа выполняет следующие функции:

1) организация ввода исходных данных в соответствии с формой их представления и видом применяемого носителя информации;

2) редактирование и организация исходных данных с целью согласования с информационными форматами, принятыми в комплекте прикладных программ;

3) определение пути обработки данных в соответствии с заказом;

4) контроль синтаксической и семантической правильности заказа на обработку данных;

5) управление процессом обработки данных по цепям типовых программ;

6) организация вывода результатов вычислений в соответствии с заказом на обработку данных.

Первый модуль предусмотрен программой для исключения резко выделяющихся наблюдений, получения частотных распределений и характеристик одномерных рядов с проверкой нормальности распределения по каждому из столбцов матрицы исходных данных. Здесь же предусматривается выполнение работ по преобразованию столбцов исходных данных в соответствии с дополнительной информацией. Вычисление характеристик рядов и исключение резко выделяющихся наблюдений может производиться до преобразований исходных данных и после преобразований.

Во втором модуле предусматривается построение матрицы парных коэффициентов корреляции K_y и K_x, определяются коэффициенты уравнения регрессии для масштабированных и натуральных переменных, вычисляется остаточная дисперсия, критерии F_1, F_2, t_i, множественный коэффициент корреляции, коэффициенты корреляции между коэффициентами регрессии и ряд других характеристик исследуемой модели.

В данном модуле проводится целенаправленный отсев аргументов или по желанию исследователя, или автоматический отсев по t-критерию Стьюдента.

Третий модуль пакета позволяет вычислять расчетные значения

функции и их отклонения от фактических значений, строятся доверительные границы для поверхности регрессии, вычисляются оценки для отклонений, определяются показатели средней и средневзвешенной ошибок аппроксимации. В этот же модуль включена программа обработки данных степенной модели.

Все программы обработки представляются в пакете в виде процедур. Во всех модулях пакета предусмотрена возможность регулирования вывода результатов вычислений.

Пакет программ позволяет обрабатывать массивы исходных данных, включающих в процедуру регрессионного анализа 63-го столбца (максимальное количество аргументов — 62) при максимальном количестве наблюдений — 2048.

В состав пакета включены также вспомогательные программы обслуживания вывода результатов расчета на различные внешние устройства, программы дублирования пакета программ на различные носители информации и программа контроля правильности хранения информации во внешней памяти.

22.3 Описание процедур пакета

Рассмотрим алгоритмы основных процедур пакета, представленные в языке АЛГОЛ-60.

Процедуры пакета разработаны в предположении, что матрица исходных данных (наблюдений) расположена в памяти машины по столбцам

$$\begin{array}{ccccc} y_1 & x_{11} & x_{12} & \ldots & x_{1P} \\ y_2 & x_{21} & x_{22} & \ldots & x_{2P} \\ y_3 & x_{31} & x_{32} & \ldots & x_{3P} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ y_n & x_{n1} & x_{n2} & \ldots & x_{nP} \end{array}$$

Предварительная обработка наблюдений предусматривает выполнение следующих условий: подготовку исходных данных к работе с модулями пакета и исключение резко выделяющихся наблюдений с последующим преобразованием матрицы исходных данных.

Матрица исходных данных может быть подготовлена к вводу по строкам. В этом случае перед работой программ пакета необходимо выполнить процедуру REBUILD1, перестраивающую исходную матрицу к виду «по столбцам».

```
procedure REBUILD1 (A, B, N, P); value N, P; integer N, P; array A, B;
begin integer I, J, K; K := 1; for J := 1 step 1 until P do
      for I := 1 step 1 until N do
      begin B[K] := A[I, J]; K := K + 1
      end
end;
```

Возможно задание исходных данных по столбцам с контрольной суммой по каждому из столбцов с целью последующей проверки правильности подготовки информации. Действия по проверке контрольных сумм и перестройке исходного массива выполняются процедурой REBUILD2.

```
procedure REBUILD2 (A, B, N, P); value N, P; integer N, P; array A, B;
begin integer I, J, K, R, H; real C; R := 0; for J := 1 step 1 until P do
      begin C := 0; for I := 1 step 1 until N do C := C + A [I, J];
            if C ≠ A [N + 1, J] then
            begin for I := 1 step 1 until N do PRINT (A [I, J]);
                  K := N + 1; PRINT (A [K, J], C); R := 1
            end
      end;
      if R = 1 then STOP (0, 1); H := 0; K := 1;
      for J := 1 step 1 until P do
      begin for I := 1 step 1 until N do
            begin B [K] := A [I + H, J]; K := K + 1
            end; H := H + 1
      end
end;
```

В случае несовпадения суммы, отперфорированной после столбца, и суммы, полученной программой, производится печать этого столбца и двух сумм. При отсутствии ошибок в подготовке данных выполняется перестройка матрицы исходных данных в соответствии с принятым форматом.

Если массив исходных данных построен по строкам с контрольной суммой на каждую строку, то необходимо выполнить процедуру REBUILD3.

```
procedure REBUILD3 (A, B, N, P); value N, P; integer N, P; array A, B;
begin integer I, J, K, R; real C; R := 0; for I := 1 step 1 until N do
      begin C := 0; for J := 1 step 1 until P do C := C + A [I, J];
            if C ≠ A [I, P + 1] then
            begin for J := 1 step 1 until P do PRINT (A [I, J]);
                  K := P + 1; PRINT (A [I, K], C); R := 1
            end
      end; if R = 1 then STOP (0, 1); K := 1;
      for J := 1 step 1 until P do for I := 1 step 1 until N do
      begin B [K] := A [I, J]; K := K + 1
      end
end;
```

После выполнения перестройки с целью получения заданного информационного формата матрица данных готова для обработки.

Перед тем как приступить к обработке статистических данных, полученных экспериментально, необходимо убедиться в том, что матрица x не содержит грубых ошибок, которые могли бы привести к существенному искажению результатов.

Просматривая столбцы матрицы x, можно обнаружить наблюдения, резко выделяющиеся по своему значению среди других элементов вектора-столбца. Иногда такие наблюдения могут быть предусмотрены физической сущностью явления, и их исключение нежелательно с профессиональной точки зрения исследователя.

В ряде случаев нельзя утверждать заранее, следует ли считать

резко выделяющееся наблюдение ошибкой без учета взаимосвязей этой переменной с другими переменными (факторами) модели. Окончательное заключение по качеству каждого из факторов модели можно дать на основании анализа всех результатов обработки.

При предварительной обработке можно использовать для принятия решения об исключении резко выделяющихся наблюдений один из известных статистических критериев, основанных на распределении кратных членов вариационного ряда в случайной выборке, извлеченной из нормальной генеральной совокупности.

Несмотря на то, что нормальное распределение допускает появление сколь угодно больших отклонений от среднего значения, вероятность появления их в выборке ограниченного объема очень мала. Поэтому, задаваясь достаточно малой вероятностью q совершить ошибку, исключая наблюдения, которые на самом деле являются верными, можно построить критические границы для крайних членов вариационного ряда в зависимости от объема выборки n. Вероятность q принято называть уровнем значимости, а вероятность $p=1-q$ — доверительной вероятностью, коэффициентом доверия или коэффициентом надежности.

Применение того или иного критерия в каждом конкретном случае зависит от полноты априорных сведений о генеральной совокупности, на основании которой получена выборка. Наиболее общей является ситуация, когда исследователь не располагает дополнительными сведениями о генеральной совокупности, кроме тех, которыми обладает матрица исходных данных X. В этом случае для оценки резко выделяющихся наблюдений каждого столбца исходной матрицы можно воспользоваться табличными значениями распределения:

$$(\overline{X}_j, S) = \max_i \frac{|X_{ij} - \overline{X}_j|}{S_j},$$

где

$$\overline{X}_j = \frac{1}{n} \sum_{i=1}^{n} X_{ij}$$

и

$$S_j^2 = \frac{1}{n} \sum_{i=1}^{n} (X_{ij} - \overline{X}_j)^2$$

при

$$j = 1, 2, \ldots, k$$

есть выборочное среднее и выборная дисперсия.

Алгоритм автоматического исключения резко выделяющихся наблюдений из матрицы исходных данных состоит в следующем. Для каждого из столбцов матрицы ошибки выявляются отдельно. Прежде всего вычисляются значения X и S_i. Затем выполняется поиск того наблюдения X_{ij}, для которого величина $\frac{|X_{ij} - X_j|}{S_j}$

является наибольшей. Если полученная величина больше значения $z_{q,n}$, заданного на основании таблицы распределения величины $(\overline{X}, S)$, то запоминается соответствующее значение i наблюдения X_{ij}. Для оставшихся $k-1$ наблюдений исследуемого столбца снова вычисляется $\overline{X}_j$ и S_j. Процедура запоминания элементов вектора-столбца повторяется до тех пор, пока не будут найдены все наблюдения, выходящие за установленные границы.

```
procedure DAIT (A, N, P, Z, K); value N, P; integer N, P, K; value Z;
real Z; array A;
begin array A1 [1 : N]; boolean array B; integer I, J; real R, X1, X2;
      for I : = 1 step 1 until N do B [I] : = false;
      for J : = 1 step 1 until P do
      begin X1 : = X2 : = 0; for I : = 1 step 1 until N do X1 : = X1 + A [I, J];
            X1 : = X1/N; for I : = 1 step 1 until N do X2 : = X2 + A [I, J] —
               X1 ↑ 2;
            X2 : = X2/N; for I : = 1 step 1 until N do A1 [I] : = 0;
            for I : = 1 step I until N do
            begin if B [I] then go to M1; A1 [I] : = ABS (A [I, J] — X2)/X2;
        M1 : end; M2 : R : = — 999999999; for I : = 1 step 1 until N do
            if A1 [I] > R then
            begin R : = A [I]; K: = I
            end; if R > Z then
            begin A1 [K] : = 0; B [K] : = true; go to M2
            end
      end; K : = 0; for I : = 1 step 1 until N do
      begin if B [I] then go to M3; K : = K + 1; for J : = 1 step 1 until P do
            A [K, J] : = A [I, J]; M3:
      end
end;
```

После завершения работы по всем столбцам из матрицы исходных данных исключаются те строки, в которых были обнаружены резко выделяющиеся наблюдения.

В связи с тем, что наименьшей из возможных дисперсий обладает только нормальное распределение, для несимметричных наблюдений установленные критические границы могут значительно перекрывать диапазон допустимых значений справа или слева. В этом случае будут исключены наблюдения, находящиеся с одной стороны вариационного ряда, а наблюдения с противоположной стороны ряда останутся, несмотря на то что достоверность некоторых из них также может вызывать сомнение. Для исключения этого явления следует сначала преобразовать исходное распределение к нормальному (если это возможно), а затем выполнять описанный выше алгоритм.

Анализ регрессии и корреляции начинается с получения характеристик одновременных рядов распределения, расчетные зависимости для которых приведены в формулах (22.1—22.15). Действия по вычислению характеристик реализуются процедурой SERIES.

```
procedure SERIES (A, P, N, Y, Y1, Y2, Y3, Y4, X, X1, X2, S, S1, S2, V, V1);
value P, N; integer P, N; array A, Y, Y1, Y2, Y3, Y4, X, X1, X2, S, S1, S2, V, V1;
begin integer I, J; real T; for J : = 1 step 1 until P + 1 do
```

```
    begin Y[J] := 0; for I := 1 step 1 until N do Y[J] := Y[J] + A[I, J];
          Y1[J] := Y[J]/N
    end; for J := 1 step 1 until P + 1 do
    begin X1[J] := 0; for I := 1 step 1 until N do
          X1[J] := X1[J] + (A[I, J] — Y1[J]) ↑ 2; X2[J] := X1[J]/N;
          X[J] := SQRT (X2[J]);      S2[J] := X1[J]/(N — 1);      S[J] :=
              SQRT (S2[J]);
          S1[J] := S[J] / SQRT (N);   INTERPOLATION (N, T); Y2[J] :=
          T × S1[J];
          Y3[J] := Y1[J] — Y2[J]; Y4[J] := Y1[J] + Y2[J]; V[J] := S[J] / Y1
          [J];
          V1[J] := V[J] / SQRT (N) × SQRT (.5 + (V[J]) ↑ 2
    end
end;
```

Близость фактического распределения к нормальному можно проверить на основании критерия Пирсона по соответствующей процедуре.

```
procedure PIRSON (A, N, X, K); value N; integer N, K; array A; real X;
begin integer I, J; real D, E, P, H, L, M, P1, S; array T[1 : D]; integer
    array B, Q[1 : D]; P1 := 3.1416; D := LN (N) / LN (2); E := 999999999;
    P := — 999999999; S := 0; for I := 1 step 1 until N do
    begin S := S + A[I]; if A[I] < E then E := A[I]; if A[I] > P then
    P := A[I];
    end; H := (P — E) / (D + 1); L := S/N; S: = 0; for I: = 1 step 1 until
    N do
    S := S + (A[I] — L) ↑ 2; P := SQRT (S/(N — 1)); for J := 1 step 1 un-
    til D do
    B[J] := 0; for I := 1 step 1 until N do
    begin M := 0; for J := 1 step 1 until D do
          begin if E + M < A[I] ∧ A[I] < E + M + H then go to M1; M :=
              M + H
          end; M1 : B[J] := B[J] + 1
    end; M := 0; for J := 1 step 1 until D do
    begin T[J] := E + H/2 + M; M := M + H
    end; for J := 1 step 1 until D do
    Q[J] := H × N/P/ (SQRT (2 × P1)) × EXP (— (T[J] — L) ↑ 2/2/P); X := 0;
    for J := 1 step 1 until D do X := X + (B[J] — Q[J]) ↑ 2/Q[J]; K := N — 2
end;
```

После расчета характеристик одномерных рядов и проверки нормальности распределения можно получить уравнение регрессии, которое содержало бы одну зависимую переменную y и p аргументов (факториальных признаков). Вместо абсолютных значений вычисления ведутся в отклонениях от арифметических средних каждой переменной, что позволяет уменьшить размер матрицы системы за счет исключения параметра b_0.

Составление и решение системы нормальных уравнений выполняется при помощи процедуры MATRICA, а обращение матрицы — при помощи процедуры INVERT.

После решения системы вычисляются по формулам 22.47—22.50 расчетные значения результативного признака, сумма квадратов отклонений, эмпирическая остаточная дисперсия, среднее квадратическое отклонение без учета потерь степеней свободы, квадрат

ошибки оценки по уравнению и ошибка оценки по уравнению множественной регрессии, несмещенная оценка дисперсии среднего значения результатного признака и ряд других показателей.

Следующим этапом анализа регрессии и корреляции является вычисление коэффициентов парной корреляции, их ошибок, критерия значимости и доверительных интервалов для коэффициентов. Эти действия выполняются процедурой:

```
procedure TWINCOR (A, N, P, Y1, Z0, Z2, R3, SR, TR1, TR2, Z12, Z21, RKM1,
    RKM2);
value N, P; integer N, P; array A, Y1, Z0, Z2, R3, SR, TR1, TR2, Z12, Z21, RKM1,
    RKM2;
begin integer I, J, K; for J := 1 step 1 until P + 1 do
      for I := 1 step 1 until N do Z0[I, J] := A[I, J] — Y1[J];
      for J := 1 step 1 until P + 1 do for K := 1 step 1 until P + 1 do
      begin Z2[J, K] := 0; for I := 1 step 1 until N do
            Z2[J, K] := Z2[J, K] + Z0[I, J] × Z0[J, K]
      end; for J := 1 step 1 until P + 1 do for K := J step 1 until P + 1 do
      begin R3[J, K] := Z2[J, K]/SQRT (Z2[J, J] × Z2[K, K]);
            SR[J, K] := (1 — R3[J, K])/SQRT (N — 1);
            TR1[J, K] := R3[J, K] / SR[J, K];
            TR2[J, K] := R3[J, K] / SQRT (1 — R3[J, K] ↑ 2) SQRT (N — 2);
            Z12 := .5 × LN (1 + R3[J, K])/(1 —R3[J, K]) — R3[J, K]/2/(N —1) —
                1.96/SQRT (N — 3); Z21 := .5 × LN (1 + R3[J, K])/(1 —
                R3[J, K]) — R3[J, K]/2/(N — 1) + 1.96/SQRT (N — 3);
                RKM1[J, K] := (EXP (2 × Z12) — 1)/(EXP (2 × Z12) + 1);
                RKM2[J, K] := (EXP (2 × Z21) — 1)/(EXP (2 × Z21) + 1)
      end
end;
```

Далее необходимо рассчитать коэффициенты частной корреляции и частной детерминации, а также их оценки. Эти действия выполняются процедурой:

```
procedure PARTCOR (A, P, N, C1, Q, RYK, RYK2, SR, TR3, TR4, Y1, E, EO);
value P, N; integer P, N; real E, EO; array A, C1, Q, RYK, RYK2, SR, TR3,
    TR4, Y1;
begin integer I, J, K, L; real ZY; ZY := 0; for I := 1 step 1 until N do
      ZY := ZY + (A[I, 1] — Y1[1]); L := 2; for J := 2 step 1 until P + 1 do
      begin MATRICA (A, C1, Q, P, L); EO[J] := ZY;
            for K := 2 step 1 until P + 1 do
            begin if K = L then go to M1; EO[J] := EO[J] — C1[K] × Z[1, K];
        M1 : end; L := L + 1
      end; for J := 2 step 1 until P + 1 do
      begin RYK2[J] := 1 — E/EO[J]; RYK[J] := SQRT (RYK2[J]);
            SR[J] := (1 — RYK2[J])/SQRT (N — P); TR3[J] := RYK[J]/SR[J];
            TR4[J] := RYK[J]/SQRT (1 — RYK2[J]) × SQRT (N — P — 1)
      end
end;
```

Коэффициенты ковариации, а также параметры доверительных интервалов для расчетных значений результатного признака (стандартные ошибки расчетных значений и концы доверительных интервалов) рассчитываются процедурой COVARIATION. При этом используется процедура INTERPOLATION для определения табличных значений параметра t, соответствующих данному числу степеней свободы.

```
    procedure COVARIATION (A, N, P, Q, COV, Y1, Y2, SY, S4, S6, Y5, Y6, Y7);
    value N, P; integer N, P; array A, Q, COV, Y1, Y2, SY, Y5, Y6, Y7; real S4, S6;
    begin integer I, J, K; real T1; array Z0[1 : N, 1 : P + 1];
          INTERPOLATION (N, T1); for I : =2 step 1 until P + 1 do
          for J : = 2 step 1 until P + 1 do COV[I, J] : = S4 × Q[I, J];
          if P > 8 then go to M1; for J : = 1 step 1 until P + 1 do
          for I : = 1 step 1 until N do Z0[I, J] : = A[I, J] — Y1[J];
          for I : = 1 step 1 until N do
          begin SY[I] : = S6; for J : = 2 step 1 until P + 1 do
                for K : = 2 step 1 until P + 1 do
                SY[I] : = SY[I] + COV[J, K] × Z0[I, J] × Z0[I, K]
          end; for I : = 1 step 1 until N do
          begin Y5[I] : = T1 × SY[I]; Y6[I] : = Y2[I] — Y5[I];
                Y7[I] : = Y2[I] + Y5[I]
          end;
M1 : end;
```

ВОПРОСЫ К ГЛАВЕ

1. Назначение пакета программ математической обработки результатов наблюдений.
2. Перечислите основные функции пакета программ.
3. В чем состоит алгоритм исключения резко выделяющихся наблюдений?
4. Запишите на алгоритмическом языке алгоритм вычисления характеристик одномерных рядов распределения.
5. Назначение и состав управляющих операторов пакета.
6. Дайте краткую характеристику пакета программ.
7. В чем состоит основной принцип конструкции пакета программ обработки наблюдений?
8. Назначение и состав процедур пакета программ.

ЛИТЕРАТУРА

Венецкий И. Г., Кильдышев Г. С. Основы теории вероятностей и математической статистики. М., «Статистика», 1968.

Гасс С. Линейное программирование. М., Физматгиз, 1961.

Езекиел М., Фокс К. А. Методы анализа корреляции и регрессии. М., «Статистика», 1966.

Лукомский Я. И. Теория корреляции и ее применение к анализу производства. М., Госстатиздат, 1961.

Пустыльник Е. И. Статистические методы анализа и обработки наблюдений. М., «Наука», 1968.

Системы сетевого планирования и управления. М., «Мир», 1965.

Смирнов Н. В., Дунин-Барковский И. В. Курс теории вероятностей и математической статистики для технических приложений. М., «Наука», 1969.

Уилкс С. Математическая статистика. М., «Наука», 1967.

Фишер Р. А. Статистические методы для исследователей. М., Госстатиздат, 1961

Худсон Д. Статистика для физиков. М., «Мир», 1967.

Яноши Л. Теория и практика обработки результатов измерений. М., «Мир», 1968.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Ж

З

И

К

Л

М

Н

О

П

Р

С

Т

У

Ф

Х

Ц

Ч

Ш

Э

Я

ОГЛАВЛЕНИЕ

Зоя Васильевна Алферова,
Галина Николаевна Лихачева,
Виктор Владимирович Шураков

«МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ЭВМ»

Редактор *Л. Д. Григорьева*
Техн. редактор *Г. А. Сидорова* Корректор *Г. А. Башарина*
Худ. редактор *Т. В. Стихно*
Переплет художника *Т. Н. Погореловой*

Сдано в набор 11/XII 1973 г. Подписано к печати 20/V 1974 г.
Формат бумаги 60×90/16 Бумага № 3 Объем 24 печ. л.
Уч.-изд. л. 26,26 Тираж 20000 экз.
А 03196 (Тематич. план 1974 г. № 87)

Издательство **«Статистика»**, Москва, ул. Кирова, 39

Великолукская городская типография управления издательств, полиграфии и книжной торговли Псковского облисполкома, г. Великие Луки, Половская, 13
Заказ № 390 Цена 1 р. 03 к.

1 р. 03 к.

СТАТИСТИКА · МОСКВА · 1974