В. А. ВИШНЯКОВ

ОРГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ ЭВМ И СИСТЕМ

Допущено Министерством
народного образования БССР
в качестве учебного пособия
для студентов вузов, обучающихся
по специальностям "Вычислительные
машины, комплексы, системы и сети" и
"Программное обеспечение вычислительной
техники и автоматизированных систем"

Минск "Вышэйшая школа" 1988

ББК 32.973я73
В55
УДК 681.3(075.8)

Р е ц е н з е н т ы: кафедра вычислительной техники Ленинградской лесотехнической академии, кафедра вычислительной техники Минского высшего инженерного зенитного ракетного училища противовоздушной обороны и профессор кафедры ЭВМ Московского высшего технического училища им. Н.Э.Баумана, доктор технических наук *В.Д.Курганов*

Вишняков В.А.

В55 Организация вычислительных процессов ЭВМ и систем: Учеб. пособие для вузов. – Мн.: Выш. шк., 1988. – 207с.: ил.

ISSN 5-339-00023-0.

Рассматриваются понятия организации вычислительных процессов в ЕС ЭВМ, СМ ЭВМ и микроЭВМ, особенности систем с разделением времени и телеобработки данных, параллельная обработка данных в многомашинных и мультипроцессорных системах. Приводятся сведения об автоматизации проектирования вычислительных процессов в ЭВМ и средствах моделирования. Указываются направления развития организации вычислительных процессов в ЭВМ пятого поколения.

Для студентов специальностей "Вычислительные машины, комплексы, системы и сети" и "Программное обеспечение вычислительной техники и автоматизированных систем".

В $\frac{2405000000-117}{M304(03)-88}$ 24–88

ББК 32.973я73

ISBN 5-339-00023-0

ПРЕДИСЛОВИЕ

Постановления партии и правительства в области вычислительной техники и решения XXVII съезда КПСС, направленные на ускорение научно-технического прогресса, предъявляют все более высокие требования по подготовке специалистов в области ЭВМ и систем.

В пособии прослеживается путь развития системного программного обеспечения ЭВМ, даются основные понятия процесса и ресурса, их распределение и управление, рассматриваются вопросы управления памятью, процессорами, устройствами ввода-вывода, организации процессов в ЕС ЭВМ, уделяется внимание организации данных, методам доступа, построению баз данных. Раскрыты важные направления развития организации процессов в ЕС ЭВМ: система с разделением времени, телеобработка данных и система виртуальных машин. Обсуждается параллельная обработка информации, организация вычислений в многомашинных, многопроцессорных и диалоговых системах. Некоторые аспекты организации вычислительных процессов в мини- и микроЭВМ рассматриваются на примерах построения операционных систем (ОС) СМ ЭВМ, а также СР/М, МР/М, UNIX. В пособии освещены вопросы автоматизации проектирования вычислительных процессов на базе аналитических, имитационных моделей и сетей Петри, а также даны понятия экспертных систем, интеллектуальных интерфейсов, баз знаний и систем логических выводов.

Поскольку такой широкий круг вопросов в одном пособии рассматривается практически впервые, некоторые из них изложены конспективно, и за более подробными сведениями читатель может обратиться к литературе, приведенной в конце книги.

Подбор материала и степень детализации изложения определяются тем, что студенты уже знакомы с основами построения ЭВМ, программированием, системным программированием, проектированием ЭВМ и систем, моделированием, мультипроцессорными системами. При изучении § 2.4–2.6, 3.3, 4.5 необходимо знание ассемблера ЕС ЭВМ. При написании § 3.5, 3.6, 6.2–6.4 использованы исследования, проводимые автором совместно с Н.И. Кузьмицким и П.З. Барасом, § 7.4 – с О.В. Германом.

Автор выражает признательность рецензентам: коллективу кафедры вычислительной техники Ленинградской лесотехнической академии (зав. кафедрой – доктор технических наук, профессор А.М. Половко), коллективу кафедры вычислительной техники МВИЗРУ (зав. кафедрой – доктор технических наук, профессор В.А. Мищенко), а также доктору технических наук, профессору кафедры ЭВМ МВТУ им. Баумана В.Д.Курганову, которые сделали ряд полезных замечаний. Автор благодарит всех, кто помогал в техническом оформлении рукописи.

Все отзывы и пожелания просьба направлять по адресу: 220048, Минск, проспект Машерова, 11, издательство "Вышэйшая школа".

Автор

СПИСОК СОКРАЩЕНИЙ

АПД – аппаратура передачи данных
АРМ – автоматизированное рабочее место
БД – база данных
БТМД – базовый телекоммуникационный метод доступа
ВМ – виртуальная машина
ВП – виртуальная память
ВПр – вычислительный процесс
ВС – вычислительная система
ВТ – виртуальный терминал
ДС – диалоговая система
ИИ – искусственный интеллект
МВК – многопроцессорный вычислительный комплекс
МВМ – монитор виртуальных машин
МД – магнитный диск
МК – макрокоманда
МЛ – магнитная лента
МПД – мультиплексор передачи данных
НД – набор данных
ОВП – организация вычислительных процессов
ОП – основная память
ОС – операционная система
ОТМД – общий телекоммуникационный метод доступа
ПДО – программа диалоговой обработки
ПЗУ – постоянное запоминающее устройство
ПО – программное обеспечение
ПОД – первичная обработка данных
ПП – прикладная программа
ППП – пакет прикладных программ
ПУ – периферийные устройства
ПУВ – программа управления выводом
ПУС – программа управления сообщениями
РВ – разделение времени
САПР – система автоматизированного проектирования
СВМ – система виртуальных машин
СМО – система массового обслуживания
СРВ – система разделения времени
ССП – слово состояния программы
СТД – система телеобработки данных
СУБД – система управления базой данных
УУ – устройство управления
ЭС – экспертная система
ЯУЗ – язык управления заданиями

ВВЕДЕНИЕ

Цель и задачи пособия. Целью данной книги является изучение программного обеспечения для организации вычислительных процессов ЭВМ в системах третьего и четвертого поколений. Вычислительные процессы рассматриваются на внешнем (пользовательском) уровне и на уровне операционной системы (ОС) – логическом. На уровне аппаратных средств организация вычислительных процессов дается в курсе "Теория и проектирование ЭВМ и систем", поэтому данные вопросы освещены в той мере, в которой это необходимо для понимания вычислительных процессов ядра ОС. Четких границ между внешним, логическим и внутренним (аппаратным) уровнями в различных ЭВМ не существует.

Основными задачами книги являются: показать на основе краткого анализа организации ЭВМ и систем состояние и перспективу развития системных программных средств; ознакомить читателя с различными средствами организации вычислительных процессов в основных семействах ЭВМ (ЕС ЭВМ, СМ ЭВМ, микроЭВМ); сформировать знания для проектирования вычислительных процессов; показать направления в разработке ЭВМ пятого поколения.

Логически пособие построено следующим образом. Во введении рассматривается исторический опыт развития средств организации вычислительных процессов (ОВП) на примерах ЭВМ четырех поколений. Далее даются основные понятия ОВП ЭВМ и систем (гл. 1), которые конкретизируются на примере организации базовой ОС ЕС ЭВМ (гл. 2, 3). В качестве средств расширения ОВП в базовой ОС ЕС ЭВМ показаны системы с разделением времени и телеобработки данных (гл. 4). Организация работ пользователя с несколькими ОС на одной ЭВМ анализируется на примере системы виртуальных машин (гл. 5). Повышение уровня работы пользователя и ОВП рассматривается на базе диалоговой системы ПРИМУС и ее расширения прикладными диалоговыми подсистемами (гл. 6). Аспекты организации процессов в многомашинных и многопроцессорных системах даются на примере ЕС и комплексов "Эльбрус" (гл. 7), а ОВП в мини- и микроЭВМ – отдельных ОС СМ ЭВМ и персональных ЭВМ (гл. 8). Задачи и средства автоматизации проектирования вычислительных процессов рассмотрены на основе аналитических, имитационных моделей и средств моделирования (гл. 9). Перспективы развития ОВП раскрываются на базе основных концепций ЭВМ пятого поколения (гл. 10). Связь между главами пособия приведена на рис. В. 1.

Материал пособия базируется на многих понятиях, которые студенты получили при изучении ряда предыдущих дисциплин: "Программирование", "Системное программирование", "Моделирование", "Теория и проектирование ЭВМ и систем", "Мультимикропроцессорные системы".

Для лучшей методической проработки материала в конце каждого раздела даются выводы, которые рекомендуется увязать с предыдущим материа-

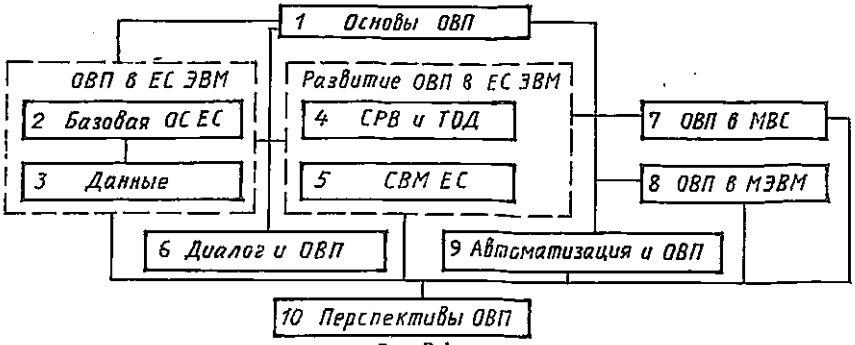


Рис. В.1.

лом, а также вопросы для контроля.

Начало развития организации вычислительных процессов в ЭВМ. В период первого поколения ЭВМ, основным назначением которых были научные расчеты, программное обеспечение базировалось на использовании машинных кодов и простейших ассемблеров. Программист работал за пультом ЭВМ, мог использовать библиотеки стандартных и служебных программ. Вычислительный процесс был полностью последовательным: ввод программы, вычисления и вывод результатов.

С изменением элементной базы для ЭВМ второго поколения расширяются как программные средства, так и области их применения. Появляются компиляторы таких языков программирования, как ФОРТРАН (1956), КОБОЛ (1959), АЛГОЛ (1960) и ПЛ/1 (1961). Для объединения модулей, транслируемых с различных языков, начинают использоваться редакторы связей, а для помещения в основную память отредактированного модуля – загрузчики.

Программисты для работы с этими системными программами применяются элементы языков управления заданиями. Для повышения скорости ввода-вывода информация записывается на устройства внешней памяти – магнитные ленты и барабаны, откуда она быстро вводится в ЭВМ и выводится на печатающие устройства. Частично операции ввода-вывода начинают выполняться параллельно с операциями обработки в процессоре. Создаются файлы последовательного, индексно-последовательного и прямого методов доступа.

Период ЭВМ третьего поколения. В конце 60-х годов появляются семейства совместимых моделей ЭВМ, имеющих разную производительность, но единые принципы построения, программную, информационную и конструкторско-технологическую совместимость. За рубежом это семейство фирмы IBM, в нашей стране в рамках содружества с социалистическими странами – ЭВМ единой системы, которые рассматриваются уже как вычислительные системы.

Значительное повышение производительности ЭВМ было достигнуто за счет введения мультипрограммирования. В таком режиме в оперативной памяти одновременно помещаются несколько программ, которые совместно используют ресурсы центрального процессора, оперативной памяти, внешних устройств и программного обеспечения. Это поддерживается механизмом прерываний, когда процессор обрабатывает одну программу, а остальные ожидают завершения ввода-вывода либо освобождения процессора.

Дальнейшее совершенствование мультипрограммирования – появление программных средств, выделяющих каждой программе фиксированный или переменный квант времени (системы с разделением времени). Эта проблема частично решалась путем распределения заданий на выполнение в несколько очередей, размещаемых на диске в соответствии с важностью (классом) задачи. К функциям ОС добавляется ввод и занесение задания в очередь, а потом загрузка его в память и выполнение. Для каждого класса задания определяется раздел оперативной памяти, загрузка очередного задания осуществляется по мере освобождения раздела. В пределах класса могут находиться задания с различными приоритетами, причем они понижаются к концу очереди.

Задание может включать вызов различных программ: трансляторов, редактора связей, программ сортировки, прикладных программ, которые должны выполняться в заданной последовательности. При этом отдельная программа составляет шаг задания.

Методы доступа и библиотечные функции остаются практически теми же, что и для ЭВМ второго поколения, за исключением введения библиотек для размещения операционной системы.

ЭВМ четвертого поколения. С развитием элементной базы появились интегральные схемы большой и сверхбольшой степени интеграции (БИС и СБИС), что еще более усугубило разницу между быстродействием процессоров и устройств ввода-вывода. Дальнейшее расширение степени мультипрограммирования ограничивалось объемом оперативной памяти (хотя она тоже стала строиться на БИС и СБИС). Увеличение объема было достигнуто введением виртуальной памяти. Если процессор в данный момент времени может выполнять одну команду, то нет необходимости размещать всю программу в оперативной памяти. Поэтому в системе с виртуальной памятью программа разбивается на сегменты (страницы) и хранится на диске. В оперативной памяти находятся активные страницы программы, пассивные страницы хранятся во внешней памяти и по мере необходимости вызываются в основную память.

Другая характерная особенность ЭВМ этого поколения связана с повышением роли данных как основного вида ресурсов. Появились централизованным образом организованные данные – базы данных и программные средства управления базами данных.

На основе баз данных строятся вопросно-ответные системы. Поскольку работа с такими системами начинает выполняться с терминалов, то развивается программное обеспечение диалогового взаимодействия, а также программная поддержка средств передачи данных. Сочетание средств управления базой данных и передачи данных используются в системах сбора информации.

В середине 70-х годов появляются семейства мини- и микроЭВМ, для которых характерно выполнение не только функций обработки данных, но в большей степени функций автоматизации проектирования и управления автоматизацией производственных процессов. Развивается программное обеспечение для выполнения таких задач, операционные системы этих ЭВМ, а также пакеты моделирования процессов, поддержки машинной графики, телеобработки, управления файловыми системами и базами данных. В конце 70-х годов появились персональные микроЭВМ, что в свою очередь стимулировало развитие программного обеспечения, языковых и информационных средств.

1. ОСНОВНЫЕ ПОНЯТИЯ ОРГАНИЗАЦИИ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

1.1. ОРГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ МАШИН И СИСТЕМ

Вычислительная система (ВС). Система – это совокупность связанных элементов, объединенных в целое для достижения определенной цели. Структура системы – это фиксированная совокупность элементов и связей между ними. Система, состоящая из одной или нескольких ЭВМ и программного обеспечения, называется *вычислительной системой*. На аппаратурном уровне ВС включает один или несколько процессоров, адресуемую оперативную и постоянную память и устройства ввода-вывода, которые связаны с процессором через каналы, или контроллеры. Каналы принимают команды ввода-вывода от процессора и посылают приказы в периферийное устройство. Процессор выполняет программу, хранящуюся в основной памяти, путем выборки и реализации команд (микрокоманд). Когда программе требуется ввод-вывод, происходит обращение к каналу, запускается канальная программа, по которой начинает работать внешнее устройство. После завершения операции ввода-вывода канал сообщает об этом процессору сигналом прерывания.

Понятия операционной системы. Вышеописанные действия происходят под управлением ОС. ОС является системой программ, предназначенных для обеспечения определенного уровня эффективности ВС за счет автоматизированного управления ее работой и предоставляемого пользователю определенного набора услуг (ГОСТ 19781–83). ОС включает набор средств для проектирования, отладки и выполнения программ, а также управления работой всей системы.

Основные понятия ОС – это процессы и ресурсы. Под *процессом* понимается выполнение программы или ее части процессором. *Ресурсом* является любой объект, который может распределяться внутри ВС: время процессора, основная и внешняя память, каналы и периферийные устройства, а также программы и наборы данных. Различают централизованное и децентрализованное распределение ресурсов. В первом случае распределение возлагается на программу-диспетчер, во втором программы, взаимодействуя, передают ресурс друг другу. Принципы распределения и разделения ресурсов занимают центральное место при построении ОС.

С точки зрения организации вычислительных процессов существует несколько видов ОС. Простейшей является однопрограммная ОС, в которой вычисления носят последовательный характер, а ресурсы не разделяются. ОС пакетной обработки характеризуется тем, что задания пользователей поступают на обработку в виде пакета, при этом возможно параллельное выполнение ввода, вычислений и вывода. Ресурсы тоже практически не разделяются.

ОС мультипрограммирования обеспечивает одновременное нахождение в основной памяти нескольких программ, которые последовательно используют время процессора, прерываясь на выполнение операций ввода-вывода. Осуще-

ствляется параллельная работа процессора, каналов и внешних устройств, при этом разделяются отдельные аппаратурные и программные ресурсы.

ОС разделения времени выполняет одновременное обслуживание нескольких пользователей путем выделения каждому из них кванта времени процессора. Усложняется организация вычислений и распределение ресурсов, в первую очередь времени процессора и основной памяти.

ОС реального времени работает в "линию" с управляемым объектом и имеет жесткие ограничения на время ответа. Организация вычислений определяется сигналами прерываний от объекта, для обработки которых за требуемое время могут быть выделены максимально возможные ресурсы. В противном случае работа управляемого объекта может исказиться.

Наконец, мультипроцессорная ОС управляет работой системы, включающей несколько в общем случае неоднородных процессоров: обрабатывающих, периферийных, функциональных и т. д. В такой ОС вычислительный процесс распараллеливается между процессорами, а распределение ресурсов носит наиболее сложный характер.

Понятие виртуальных машин. Одной из современных моделей ВС является кольцевая модель (рис. 1.1). В центре кольца находится аппаратура ЭВМ *0*, затем ядро ОС *1*. Состав ядра переменный, но в него включаются программы управления процессами, распределения ресурсов и обработчики прерываний. Далее следуют программы ввода-вывода (драйверы) *2*, управления файлами *3*. Уровень *4* образуют обрабатывающие программы: трансляторы, редакторы связей и загрузчики. Пользовательские программы расположены на уровне *5*, там же находятся интерфейсные программы для связи с ВС.

Необходимо различать реальную (физическую) аппаратуру ЭВМ (нулевой уровень) и машины, воспринимаемые различными классами пользователей (программистами, операторами). Последние являются *виртуальными машинами* (ВМ), поскольку они образуются за счет наслоения программного обеспечения на аппаратуру. На самом нижнем уровне находится реальная (физическая) ЭВМ. Далее системный программист, использующий систему на уровне i и ниже, пишет программное обеспечение для построения уровня $i+1$. Гра-

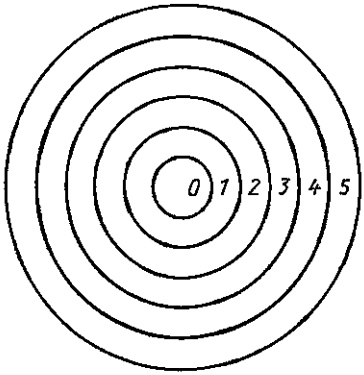


Рис. 1.1.
Кольцевая модель вычислительной системы.

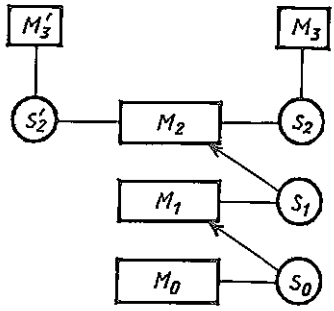


Рис. 1.2.
Иерархия виртуальных машин.

фическое изображение этой иерархии ВМ показано на рис. 1.2, где M_i обозначает машину, а S_i – программное обеспечение на i-м уровне. Программный модуль S_0, представленный на машинном языке M_0, реализует новую ВМ – M_1. Программное обеспечение S_2 для ВМ M_2 дает новую машину M_3, в то же время S'_2, написанное для M_2, дает еще одну машину M'_3.

ОС и "языковые" процессоры (трансляторы) создают виртуальные машины для пользователей, их основное назначение – транслировать машины с более высокого уровня на низкий до реальной ЭВМ.

С понятием структуры ВС связано понятие архитектуры. Этот термин может быть определен как распределение функций, реализуемых системой на отдельных уровнях, и уточнение границ. Специфическим свойством архитектуры ВС является возможность выделения в ней уровней абстракции, частично показанных на рис. 1.3. На уровне *1* система взаимодействует с внешним миром через два набора интерфейсов: языки (оператора, программиста, управления заданиями) и программы-утилиты. Уровни *2–4* определяют архитектуру программного обеспечения. На уровне *2* прикладные программы взаимодействуют с трансляторами, не предоставленными в распоряжение пользователя на уровне *1*. На уровне *3* реализуются функции по управлению базой данных, файлами, виртуальной памятью. К уровню *4* относятся функции управления внешней и оперативной памятью, другими аппаратными средствами, а также процессами системы. Уровень *5* отражает границу между системным программным обеспечением и аппаратурой.

Существуют четыре ключевые проблемы, которые связаны с функционированием ОС: преобразование виртуальных машин, распределение ресурсов, защита программ и взаимодействие процессов.

Имеется устойчивая тенденция расширения функций аппаратуры в кольцевой модели ВС (см. рис. 1.1), что связано с наличием в настоящее время се-

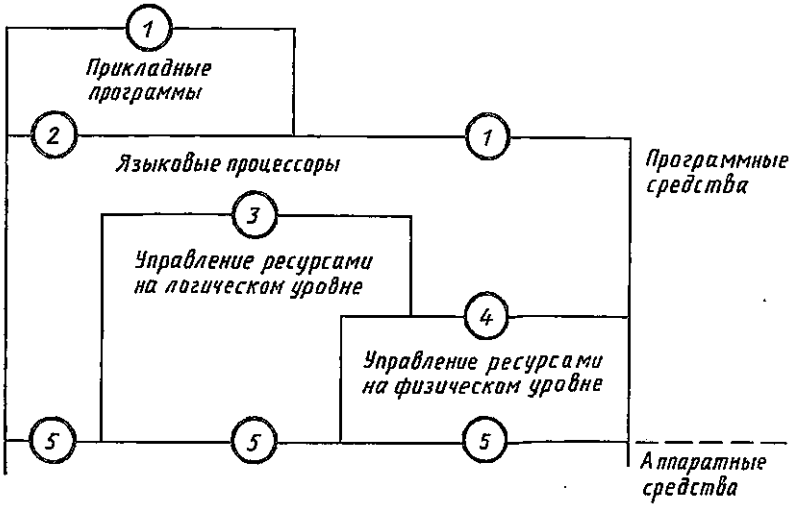


Рис. 1.3.
Многоуровневая организация архитектуры ВС.

мантического разрыва между классической структурой ЭВМ и сложностью программного обеспечения верхних уровней. Этот разрыв заполняется средствами ОС, языковыми процессорами, диалоговыми системами и т. д. Поэтому в настоящее время некоторые из функций ядра реализуются аппаратно или микропрограммно.

1.2. ВИДЫ И ФУНКЦИОНИРОВАНИЕ ОПЕРАЦИОННЫХ СИСТЕМ

Виды ОС. По принятому методу организации вычислительных процессов ОС делятся на однопрограммные, пакетные обработки, мультипрограммные, разделения времени и реального времени, а также смешанные. С точки зрения работы пользователей они обеспечивают диалоговую работу в локальном или удаленном режиме (телеобработка).

По типу используемых запоминающих устройств ОС подразделяются на перфоленточные, магнитоленточные (для микроЭВМ), дисковые и размещаемые в ПЗУ. В состав ОС входят управляющие программы, средства контроля аппаратных и программных средств (тестовое обеспечение), сервисные средства, средства разработки и отладки программ (системы программирования). Поэтому по функциональному назначению ОС можно разделить на инструментальные – только для разработки и отладки программ, исполнительные – для выполнения готовых программ и управления вычислительным процессом и общего назначения – для разработки и отладки программ, а также их выполнения.

Функционирование ОС. Работу операционной системы рассмотрим в мультипрограммном режиме как наиболее общем. Пусть ВС включает несколько обрабатывающих устройств (процессоров), модули основной памяти, процессоры ввода-вывода, контроллеры периферийных устройств, устройства внешней памяти, периферийные устройства и терминалы. Процессоры могут быть однородными или неоднородными. Удаленные терминалы могут подключаться к линии связи через контроллеры. При такой организации аппаратурные средства должны обладать рядом свойств: включать средства приоритетного прерывания, иметь защиту программ и памяти, обеспечивать динамическую настройку адресов, содержать таймер, базовые регистры и память прямого доступа.

Требования к программному обеспечению ВС рассмотрим на примере прохождения задания пользователя через систему. 1. Задание J считывается в основную память и записывается в память прямого доступа. Из спецификации задания извлекается информация для планирования (приоритет, требования к памяти, времени процессора, число и вид устройств ввода-вывода и т. д.). 2. На основании состояния текущих процессов в системе и характеристик задания J последнее выбирается для выполнения. Распределяется основная память и загружается программа задания, которая инициирует создание нового процесса j. 3. Процесс j может развиваться на процессоре P_i. Если в систему поступит более приоритетный процесс, то j может быть занесен в список готовности. После продолжения j может быть приостановлен из-за отсутствия ресурса. В общем случае j будет проходить через состояния готовности, продолжения и блокировки, эти изменения являются управляемыми по прерыванию. 4. Про-

цесс j завершился нормально или аварийно. Основная память перераспределяется, внешняя освобождается, результаты выполнения записываются на выходной файл. 5. Программа вывода в порядке очереди распечатывает результаты задания, затем J удаляется.

Из вышеизложенного можно сделать следующие выводы: процессы создаются, уничтожаются, а также меняют свое состояние; процессы нуждаются в связи друг с другом; ресурсы динамически закрепляются за процессами и освобождаются; требуется обработка ввода-вывода и файлов.

Структура программного обеспечения (ПО). Для управления такой системой ПО должно включать прежде вмего программы, обеспечивающие управление процессами: создание, удаление, связь и планирование. Второй составляющей является система управления ресурсами, которая распределяет ресурсы основной и внешней памяти, устройства ввода-вывода, процессоры и т. д. Третий компонент – это файловая система, ответственная за создание, разрушение, модификацию и восстановление информации во внешней памяти и на периферийных устройствах. Необходимость и порядок использования этих компонентов определяются четвертой составляющей ПО – программами обработки прерываний.

ОС может рассматриваться как расширение аппаратуры, т. е. как виртуальная машина для организации и управления вычислительными процессами. Та часть ОС, которая постоянно находится в основной памяти, называется *ядром*. Множество программ ядра включает базовые примитивы управления процессами, распределения ресурсов, управления вводом-выводом и обработки прерываний. Имеются тенденции аппаратной реализации части ядра.

Управление процессами. Процессы непрерывно поступают в ВС, затем покидают ее, поэтому необходимы примитивы для их создания и уничтожения. Создание включает построение вектора состояния процесса, списка требуемых ресурсов, идентификацию процесса. При уничтожении процесса его ресурсы освобождаются, а спецификации удаляются из системных наборов данных. Связь процессов реализуется через различные примитивы, рассмотренные далее, а планирование осуществляется программами-планировщиками.

Управление ресурсами. Процедуры распределения и освобождения связаны с каждым классом ресурса. Общий вид этих процедур: Распределить, Освободить с указанием класса ресурса, идентификатора и числа. Методы управления различными классами ресурсов различны.

Управление вводом-выводом. Основными примитивами являются операции: ЧИТАТЬ, ПИСАТЬ, УПРАВЛЯТЬ. В ядре возможна операция ввода-вывода: ВЫПОЛНИТЬ (канал, программа канала), которая вызывает программу ПЛАНИРОВЩИК канала. По завершении операции ввода-вывода выдается сигнал прерывания и происходит его обработка. На внешнем уровне запрос на ввод-вывод по команде ЧИТАТЬ (устройство, область) вызывает элементы файловой системы для отыскания каталога, файла, его открытия, построения буферов передачи, выхода на требуемую запись в файле, выполнения операции. Эти функции могут быть как в составе ядра, так и вне его.

Обработка прерываний. Внешние и внутренние прерывания могут рассматриваться как сигналы пробуждения блокированных процессов, сигнализирующих об одном из событий: процесс завершен – завершение аппаратных операций канала или программного процесса; затребовано обслуживание –

оператор нажал кнопку "Внимание" на клавиатуре терминала; произошла ошибка ввода-вывода, адресации, переполнения и т. д. Программный обработчик прерывания должен сохранить состояние прерванного процесса, определить, какой процесс необходимо запустить, восстановить после его завершения прерванный процесс.

Пользовательский интерфейс. Компоненты ОС непрерывно вызываются пользовательскими программами через запросы на ввод-вывод, запросы ресурсов, а также в ответ на прерывания. На более высоком уровне информация о программах и файлах связывается посредством командных и управляющих языков. Пользователи применяют их для запросов на выполнение работ, а операторы – для управления системой. Наблюдается тенденция к объединению этих языков в один. Традиционная форма предложения языка: команда <список операторов>.

Методология проектирования ОС. Основными функциями ОС являются эффективное обслуживание пользователей, управление процессами и распределение ресурсов. Для этого ОС рассматривается как динамический набор взаимодействующих процессов, которые связываются через системные структуры данных. Основными задачами при проектировании ОС являются: описание виртуальной машины пользователей, определение процессов и их взаимосвязи для реализации этой машины. Существует несколько подходов к проектированию ОС. Рассмотрим два из них, предложенные Е.Дейкстрой и Б.Хансеном.

Иерархический подход к абстрактной машине Дейкстры основан на размещении процессов ОС в иерархическом порядке. При этом каждый следующий уровень определяет более абстрактную машину. Чем он выше, тем больше выполняется задач по управлению ресурсами. На уровнях, начиная с нулевого, могут быть следующие задачи: 0 – распределение процессора; 1 – управление основной и внешней памятью; 2 – связь между виртуальными процессами и терминалом оператора; 3 – буферизация ввода-вывода и управление периферийными устройствами; 4 – программы пользователей. Процессы выше нулевого уровня "не интересуются" числом доступных физических процессоров, выше первого работают только с сегментами информации, независимо от ее размещения, и т. д. Синхронизация в системе достигается за счет аппаратного механизма прерываний и программными средствами. Например, прерывания от таймера связаны с нулевым уровнем, от ввода-вывода – с первым (при работе с внешней памятью). Данная методика разделяет функции ОС и устанавливает каналы связи между уровнями, что снижает сложность проектирования и отладки. Основные трудности связаны с определением структуры и функций уровней.

Подход Б.Хансена заключается в том, чтобы концентрировать усилия на проектировании ядра, которое универсально для многих модификаций ОС. Это связано с тем, что стратегия распределения ресурсов в ОС должна расширяться в ответ на изменение требований, появление новых приложений и способов работы.

Принципы, предложенные Дейкстрой и Хансеном, а также опыт других разработок ОС легли в основу направлений проектирования. Последовательность возможных шагов проектирования: 1) определение ВМ пользователя, разработка командного языка и описание функций, выполняемых каждой ко-

мандой; 2) описание путей прохождения заданий пользователя через аппаратуру и ОС; 3) определение процессов, необходимых для реализации п. 1, 2. Нахождение функций каждого процесса и алгоритмов его взаимодействия с другими процессами, а также структуры данных, требуемых для связи. Размещение процессов в иерархическом порядке; 4) определение компонентов ядра системы и их структур данных. Разработка стратегии распределения ресурсов; 5) обоснование правильности проектирования и определение поведения системы с использованием моделирования и аналитического аппарата.

1.3. ВЗАИМОДЕЙСТВУЮЩИЕ ПРОЦЕССЫ

Работы с процессами. Увеличение производительности ВС в настоящее время связано с усовершенствованием технологии элементной базы, но в большей степени с построением параллельных ЭВМ. Несколько процессоров (центральный, ввода-вывода, терминальный) могут быть связаны с общей памятью и управлением. В этом случае будут выполняться параллельно и обмениваться сообщениями несколько процессов.

Пусть $S(a, b)$ означает последовательную связь процессов, а $P(a, b)$ – параллельную связь процессов a и b. Граф развития процессов является правильно вложенным, если он может быть описан функциями S, P или комбинацией этих функций. Графы процессов, представленных на рис. 1.4, могут быть описаны в виде:

$S(p_1, S(p_2, S(p_3, p_4)))$ (рис. 1.4, *а*); $P(p_1, P(p_2, P(p_3, p_4)))$ (рис. 1.4, *б*);

$S(p_1, S(P(p_2, P(p_3, P(p_4, p_5)), p_6)))$ (рис. 1.4, *в*).

Примерами параллельных процессов могут быть вычисления арифметических выражений, сортировка данных, умножение матриц.

Хотя процесс – это работа, он не может быть отождествлен с программой. Скорее это пара <процессор, программа> при выполнении. Развитие процессов может быть описано как последовательность векторов состояния S_1, S_2, ..., S_i, где каждый вектор содержит указание на следующую программную команду. Вектор состояния можно рассматривать как информацию процессору для развития процесса. Взаимосвязь между процессами и управление их работой осуществляются с помощью базовых операций. Процесс является продолжающимся, если он выполняется процессором, и находится в состоянии готовности, если мог бы выполняться. Процесс блокирован, если не может вы-

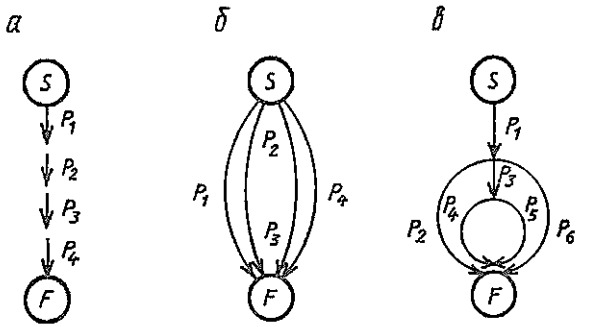


Рис. 1.4.
Графы процессов.

полняться, пока не получит необходимый ему ресурс или сообщение от другого процесса.

Есть два подхода к образованию программных процессов. При первом строится система с фиксированным числом процессов, которые существуют всегда. Чтобы выполнить некоторую работу, нужно получить в распоряжение один из программных процессов. Второй подход заключается в наличии механизма, который образует и уничтожает программные процессы по запросу. Этот механизм называется *стержнем*, он позволяет системе манипулировать процессами. При этом одни процессы могут вызывать или уничтожать другие. В стержне предусмотрены команды ЗАПУСТИТЬ и ОСТАНОВИТЬ. Первая выделяет процессу процессор, вторая отбирает. По команде УНИЧТОЖИТЬ у процесса отбираются ресурсы и стирается информация о его состоянии. Нескольким процессам стержень может выдавать по кванту времени.

Процессы могут существовать как несвязанные единицы или быть связаны отношениями, образуя структуры. Если не предусматривается структурной организации, то стержень по мере надобности (запрос пользователя, решение задач) образует процесс, а после завершения его уничтожает. В системе с древовидной структурой процесс, запускающий другие процессы, называется "отцом", а образованные процессы – "потомками". Дерево процессов в системе есть ориентированный граф, в котором каждой вершине ставится в соответствие процесс, а дугам – связи между ними. В такой системе могут действовать строгие правила передачи ресурсов и организации управления. При образовании каждый процесс может получить только те ресурсы, которые принадлежат его "отцу", и только "отец" может контролировать "потомков".

После того как процессы созданы и запущены, они могут работать независимо и обмениваться информацией. Для этого вводится аппарат связи между ними. Для повышения производительности ВС необходимо реализовать эффективное разделение ресурсов между процессами. Физические устройства (процессор, память, каналы и т. д.) образуют физические ресурсы, а программные средства и данные – логические ресурсы. Чтобы, например, разделить диск между несколькими процессами, нужно программно реализовать несколько логических (виртуальных) дисков. Процессу, использующему ресурс, неважно, является он физическим или логическим, главное, чтобы он выполнял свою функцию. Физические и логические ресурсы можно разделять, но важно, чтобы в каждый момент времени они были доступны одному процессу.

Ресурс, который допускает обслуживание пользователя за один раз, называется *критическим*. Если несколько процессов пользуются таким ресурсом, они должны синхронизировать свои действия. Место, где процесс обращается к такому ресурсу, называется *критическим участком*. В каждый момент времени только один процесс может выполнять свой критический участок при работе с одним ресурсом. С этой целью в систему вводится механизм синхронизации, который обладает двумя свойствами. Во-первых, при запросе критического ресурса несколькими процессами только один получает разрешение. Во-вторых, в каждый момент времени только один процесс должен находиться в своем критическом участке.

Пусть один процесс отправляет сообщение, а другой его получает. Одним из вариантов связи является набор буферов (пул), каждый из которых содержит одно сообщение. Первый процесс может передавать сообщения, пока есть

свободные буфера, а второй получать их, пока пул не пуст (отношение поставщик–потребитель).

Наиболее известные приемы синхронизации на нижнем уровне – блокировка памяти, семафоры, проверка и установка. Любого из них достаточно, чтобы решить проблему критических участков и отношений поставщик–потребитель. Блокировка памяти заключается в том, что если два процесса пытаются обратиться в одну ячейку памяти, то одному из них разрешается выполнить операцию, а другому приходится ждать. Это позволяет реализовать взаимное исключение двух и более процессов. Машинная операция проверки и установки решает проблему критического участка посредством блокировки памяти. Однако два вышерассмотренных приема не всегда эффективны, поскольку другие процессы могут попасть в активный цикл и напрасно расходовать процессорное время. Для предотвращения этого используются семафоры.

Семафор. Семафором называется неотрицательная целая величина, для которой введены две операции: V(S) – увеличение S на 1, P(S) – уменьшение S на 1. Если P(S) = 0, то уменьшение невозможно, и процесс, вызывающий операцию P, вынужден ждать. Операции V и P неделимы, т. е. в каждый момент времени только один процесс может выполнять операции P и V над данным семафором. С помощью семафора организуется взаимное исключение процессов заключением критических участков в скобки, роль которых выполняют P и V . Если два процесса используют семафор S , то при S = 1 ни один из них не находится в критическом участке, при S = 0 – один.

В целом с помощью общих семафоров организуется управление ресурсами. Однако в отдельных случаях семафоры неудобны из-за их примитивности. Поэтому используются приемы синхронизации на верхнем уровне. Первый из них называется *почтовым ящиком*. Если процесс P1 хочет общаться с процессом P2, он запрашивает у системы почтовый ящик. Под последним понимается структура, состоящая из заголовка и гнезд для передачи сообщений. P1 может посылать сообщения, пока не заполнены гнезда, а P2 – получать сообщения, пока гнезда не пусты. Эту простую структуру можно усложнить для передачи сообщений в обоих направлениях. Тогда в гнезде содержится либо сообщение от P1, либо ответ от P2, причем гнездо не будет занято, пока на сообщение не придет ответ. В случае, если многим процессам необходимо общаться с одним, можно организовать многовходовый почтовый ящик. Почтовые ящики удобны для передачи сообщений, но их использование требует специального процесса для контроля за нахождением одного процесса в критическом участке.

Еще одним средством синхронизации является монитор. Под *монитором* понимается набор процедур и информационных структур, которым процессы пользуются в режиме разделения, причем в каждый момент – один процесс. Примером монитора является планировщик ресурсов. Монитор может задержать процесс, если последний запрашивает занятый ресурс, до его освобождения. Это выполняется посредством операций ЖДАТЬ и СИГНАЛ. При блокировке монитор указывает условие, при котором процесс продолжается. В мониторе может находиться одна или несколько процедур, допускающих параллельную работу. Мониторы имеют преимущество перед семафорами. Они более гибки, могут реализовать, кроме семафорных, другие синхронизирующие операции, например механизм почтовых ящиков. Мониторы также дают возможность процессам совместно использовать программу, представляющую со-

бой критический участок. В целом вариантов связи между процессами гораздо больше, чем мы рассмотрели, но в принципе большинство синхронизирующих механизмов похожи либо на семафоры, либо на почтовые ящики, либо на мониторы.

Иногда при синхронизации процессов могут возникнуть тупиковые ситуации. Это положение, когда два процесса задерживают друг друга, не давая выполнить запросы на ресурсы и не желая их освободить. Например, P1 и P2 – параллельные процессы, один из них имеет ресурс V1, второй – V2, и оба запрашивают ресурсы V2 и V1 соответственно. Для разрешения тупиковых ситуаций существуют три стратегии: предотвращение, автоматическое обнаружение и ликвидация их, а также уничтожение их оператором. В первом случае при запросе процессом ресурса, который может вызвать тупиковую ситуацию, этот запрос либо не удовлетворяется, либо другой ресурс отбирается у второго процесса. Однако этот путь приводит к простою ресурсов.

Вторая стратегия заключается в программном обнаружении тупиковой ситуации, и у части процессов отбираются ресурсы, чтобы запустить другие процессы. Например, в случае с процессами P1 и P2 можно либо у P1 отобрать V1 , отдав его P2, либо наоборот.

В третьем случае считается, что тупики встречаются редко, поэтому при их возникновении оператор просто заново запускает систему. Наиболее простой алгоритм заключается в том, что каждый процесс запрашивает максимальное число ресурсов, но для этого он должен отдать все свои ресурсы.

1.4. УПРАВЛЕНИЕ ПРОЦЕССАМИ И РЕСУРСАМИ

Подход. В ответ на запросы командного языка в ОС создается пользовательский процесс P_j , который отвечает за контроль и управление прохождением задания через систему. Процесс P_j может создавать ряд процессов-потомков, каждый из которых соответствует затребованной единице работы. Эти процессы могут создавать другие процессы на различных уровнях ОС последовательно или параллельно (рис. 1.5) . Рассмотрим структуры данных и базовые операции для процессов и ресурсов, а также методы распределения процессов между процессорами, т. е. диспетчеризацию.

Структура данных для процессов и ресурсов. Каждый класс ресурсов и процессов имеет структуры данных, которые отражают их состояние. Эти структуры данных называются *дескрипторами,* определяют состояние ОС и обрабатываются системными программами. Дескриптор процесса может быть задан пятеркой вида $D_i = (I, VR, ST, P, PR)$, где i – номер процесса; I — идентификатор для внешнего обращения; VR определяет виртуальную машину и включает состояние процессора (слово состояния программы), номер обрабатывающего процессора, план основной памяти, другие ресурсы (периферийные устройства, внешнюю память, критические секции и т. д.) и список созданных ресурсов; ST включает состояние процесса (выполняе-

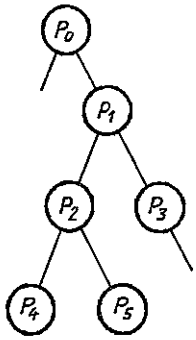


Рис. 1.5. Структура процессов задания.

мый, готов, ожидающий) и данные, связанные с состоянием (указатели на списки готовности и ожидания); P указывает на "отца", который создал процесс,и прямых "потомков"; PR – данные для учета и планирования (приоритет).

Каждый класс ресурса задается тройкой $R_i = (Id_n, S_\omega, S)$, где Id_n – число и идентификация доступных единиц ресурса; S_ω – список ожидания блокированных процессов; S – распределитель, ответственный за удовлетворение запросов; i – номер ресурса. Для каждого повторно используемого ресурса вводится семафор, который задается пятеркой вида $S_i = (Id_i, Cr_i, Av_i, W_i, Al_i)$, где Id_i – идентификатор; Cr_i – имя процесса-создателя семафора; Av_i – указатель начала списка доступности для семафора ресурса; W_i – список ожидающих процессов; Al_i – точка входа в распределитель. Для списка ожидания W_i выполняются операции вставки в очередь и удаления из нее. К распределителю можно обратиться следующим образом: $Al(r, k, L)$, где k – число процессов с распределенными ресурсами; L – их список; r – имя семафора.

Важным ресурсом является процессор, дескриптор которого может быть определен парой $D_p = (np, P_i)$, где np – число процессоров; P_i – идентификатор процесса, развивающегося на процессоре. Дескрипторов процесса и ресурса вполне достаточно для описания и работы в различных ОС. Однако немногие системы включают такой полный набор дескрипторов, поскольку универсальные структуры приводят к снижению эффективности.

Операции над процессами и ресурсами. Основные операции, определенные над процессами, могут быть заданы пятеркой вида $O_p = (Cr, Ds, Ss, Ac, Cp)$, где Cr означает создать процесс; Ds – уничтожить; Ss – приостановить; Ac – активизировать; Cp – изменить приоритет. Чтобы создать нового потомка, процесс вызывает операцию Cr с аргументами: внешнее имя, начальное состояние процесса, основная память, список других ресурсов, приоритет. При разрушении указывается имя процесса. При обращении к процедуре приостановки задается имя процесса и область возврата. Активизация процесса предполагает изменение его состояния на активное с указанием его номера и возможное обращение к планировщику. Уничтожение процесса. как и приостановка, относится ко всем потомкам процесса. Процедура Cp изменяет приоритет процесса. Во всех операциях с процессами необходимо предусмотреть проверку на ошибки, например, проверить, является ли процесс P_k потомком вызывающего.

Взаимодействие процессов и распределение ресурсов происходит посредством операций над семафорами. Четверка операций для работы с семафорами имеет вид $O_r = (C_R, D_R, R_g, R_l)$, где C_R означает создать семафор ресурса; D_R – уничтожить семафор ресурса; R_g – запросить элементы семафора ресурса; R_l – освободить их.

Процесс, запрашивающий ресурсы по команде C_R, указывает их имя, детали запроса и область памяти для запоминания. Действия команды R_g состоят в том, чтобы занести процесс в список ожидания, связанный с R, а затем вызвать распределитель. Если запрос удовлетворяется, то процесс переводится в состояние выполнения или готовности, в противном случае – в состояние ожидания. Если процессу больше не требуется ранее запрошенный ресурс, используется операция R_l с параметрами: семафор ресурса, характеристики ресурса. На рис. 1.6 представлены все возможные состояния процессов и операции, вызвавшие эти состояния.

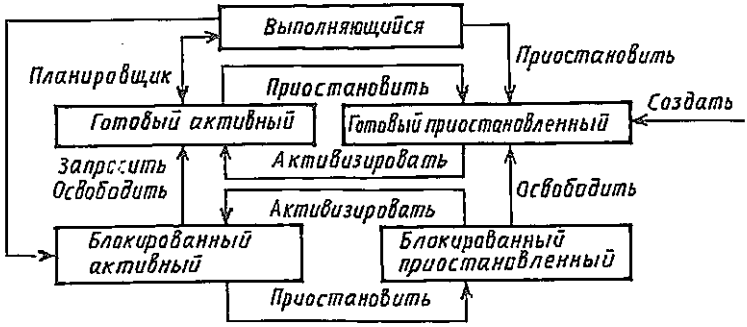


Рис. 1.6.
Состояния процессов.

Организация планировщика процессов. Планировщик (диспетчер) представляет собой часть ОС, предназначенную для распределения процессоров процессам, находящимся в состоянии готовности. Рассмотрим организацию планировщика и стратегии диспетчеризации. Новое распределение процессов происходит в случае блокировки, готовности или изменения приоритета отдельных из них, а также изменения числа процессоров.

Важным источником блокирования и готовности процессов является таймер, который в простейшем случае задает постоянные кванты времени. В более гибкой системе кванты могут вычисляться или динамически изменяться системными процессами.

Различают централизованное и децентрализованное построения планировщика (рис. 1.7). В первом случае это отдельный процесс, развивающийся на своем процессоре. Он непрерывно проверяет запросы или вызывается по заявкам. В случае децентрализованной организации планировщик разделяется процессами. Централизованная организация используется в мультипроцессорных системах. При этом процессор планировщика непрерывно опрашивает сигналы от процесса, связанного с пробуждением процессов и изменением приоритетов. Алгоритм планировщика заключается в следующем: взять готовый процесс с наивысшим приоритетом, если есть свободный процессор, распределить на него процесс, иначе дать сигнал переключения процессору с менее приоритетным процессом.

Методы планирования. Одним из простых методов является предоставление процессора процессу с наибольшим приоритетом. Если вытеснение не допускается, назначенный процесс будет выполняться, пока не завершится или не заблокируется. При поступлении в очередь более приоритетного процесса последний будет ждать, пока не освободится процессор. При разрешении вытеснения текущий процесс с меньшим приоритетом прерывается, вытесняется в очередь, а его место занимает процесс с бóльшим приоритетом. При освобождении процессора ему назначается первый процесс из очереди готовых процессов, если она упорядочена по приоритетам (алгоритм постановки прерванного процесса в такую очередь усложнен). В случае неупорядоченной очереди ее нужно просматривать при назначении на процессор, но при этом упрощается постановка в нее нового процесса.

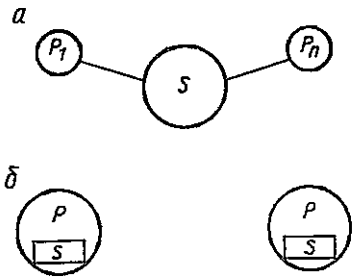


Рис. 1.7.
Построение планировщика:
а – централизованное; *б* – децентрализованное.

При планировании важно назначение приоритетов. Простейшим является назначение приоритетов, обратно пропорциональное времени выполнения процесса. При этом можно использовать известное или ожидаемое время. Поскольку информация о времени выполнения часто неточна, во многих стратегиях планирования приоритет вычисляется из длительности пребывания в системе. Приоритет возрастает с коэффициентом a во время ожидания в очереди готовых процессов и с коэффициентом b во время выполнения. В зависимости от выбора a и b возможны различные правила планирования. Если $0 < a < b$, то очередь обслуживается в порядке поступления, при $0 > b > a$ – в обратном порядке. Можно предложить несколько правил нахождения линейно возрастающего приоритета. Например, с целью обеспечения занятости внешних устройств процессам может присваиваться высокий приоритет в моменты интенсивного использования ввода-вывода.

Круговорот (Round Robin). Это правило диспетчеризации FIFO, когда для каждого процесса из очереди устанавливается фиксированный квант времени. Если процесс развивается на процессоре в течение t единиц времени, он вытесняется и помещается в конец очереди, а процессор назначается следующему процессу, но не более чем на t единиц времени. Квант может также динамически изменяться, например, $t = T/n$, где T – интервал; n – число процессов в системе. Если $t = \infty$, получаем стратегию обслуживания в порядке поступления (FIFO).

Существует много разновидностей круговорота, например величина кванта может связываться с приоритетом процесса или получаться из схемы линейно возрастающего приоритета (круговорот со смещением). В алгоритме с обратной связью используется m очередей, каждая из которых обслуживается в порядке поступления. Попадая в систему, процесс поступает в первую очередь, после обслуживания – во вторую и т. д. Процессор обслуживает непустую очередь с наибольшим приоритетом.

Основное преимущество очередей с обратной связью и круговоротом перед обслуживанием по приоритету заключается в том, что они эффективнее для коротких заданий и не требуют информации о времени выполнения процесса. Если приоритет вычисляется как функция ресурсов, выделяемых процессу, отдается предпочтение процессам с высоким приоритетом. В этом случае можно использовать смешанные правила планирования, например круговорот со смещением.

1.5. УПРАВЛЕНИЕ ОСНОВНОЙ ПАМЯТЬЮ

Подход. Основными функциями управления основной памятью являются: учет свободной и занятой памяти, принятие решения о ее распределении, выделение памяти процессу и корректировка информации о ее состоянии, освобождение памяти с корректировкой ее состояния.

Различают статические и динамические методы распределения основной памяти. При статическом методе вся необходимая память назначается до начала выполнения программы, при динамическом – во время выполнения. В первом случае связывание, перемещение и распределение памяти происходят до того, как процессы переходят в состояние готовности. Достоинством является относительная простота реализации, недостатком – отсутствие гибкости. При динамическом распределении назначение и освобождение памяти происходят во время выполнения программы.

Принцип распределения памяти называется стратегией. Простейшей стратегией является загрузка программы целиком. Однако в каждый конкретный момент времени выполняется небольшая часть программы, остальную можно держать во внешней памяти. При этой стратегии память используется более эффективно, но требуются дополнительные аппаратно-программные средства для перемещения частей программы между основной и внешней памятью. Простейшим случаем перемещения частей программы являются оверлейные структуры. В этом случае корневой модуль программы находится в основной памяти, а остальные модули загружаются по мере необходимости, перекрывая друг друга.

Однако существуют две проблемы, которые не решаются применением оверлейной структуры. Первая связана с тем, что нельзя заранее определить, какие модули нужны в памяти в конкретный момент, поэтому программист не всегда может описать оверлейную структуру. Вторая проблема заключается в том, что не всегда заранее известен объем свободной оперативной памяти. В одних случаях нет места для перекрытий модулей, в других – в перекрытиях модулей нет необходимости. Решением этих проблем является сегментация и страничная организация памяти.

Сегментация. Сегментация – это организация программы, при которой ее адресная структура разбивается на отрезки разной длины (*сегменты*) в соответствии с логической структурой программы (процедуры, блоки, области данных).

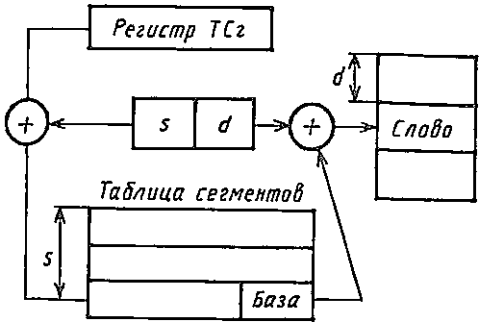


Рис. 1.8.
Вычисление адреса при сегментации.

Команда программы при сегментации задается адресом $A = (s\ .\ d)$, где s – адрес сегмента; d – смещение внутри сегмента. При расположении программы в памяти сегменты могут располагаться в ее различных участках, а отдельные сегменты – во внешней памяти. Если при обращении к сегменту его не окажется в основной памяти, возникнет прерывание. Средства ОС по обработке прерывания произведут его загрузку из внешней памяти. Хотя сегментирование программы легче поддается распределению, его реализация требует дополнительного аппаратного оборудования.

Каждому процессу (заданию) в системе назначается всегда присутствующая в памяти таблица сегментов (ТСг), в которой каждому сегменту отводится одна запись. С помощью этой таблицы производится отображение программного адреса в физический. Строка таблицы может содержать бит присутствия сегмента в основной памяти, базовый адрес сегмента, его границы (длину), бит защиты. Для обращения к любой команде необходимо с помощью регистра таблицы сегмента обратиться к самой таблице (рис. 1.8). В s-й строке таблицы указан адрес сегмента, его d-я ячейка содержит искомое слово. При этом требуются два обращения к памяти: одно – к таблице, другое – к слову внутри сегмента.

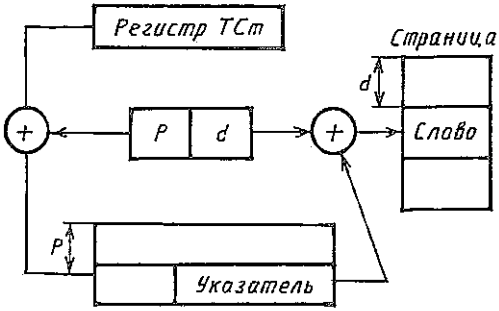
Прежде чем обратиться к вычислению адреса, система аппаратным путем проверяет признак присутствия сегмента в основной памяти. Если он есть – адрес вычисляется, если нет – происходит прерывание, которое вызывает супервизорную программу обработки таких прерываний. Эта программа отыскивает нужный сегмент во внешней памяти, проверяет наличие для него места в оперативной памяти (при отсутствии пересылает один из сегментов во внешнюю память) и загружает его. При переходе на другой процесс (программу) система заменяет содержимое таблицы сегментов.

Преимуществом сегментной организации является перекрытие программы без участия программиста. Другое преимущество связано с объединением модулей. При программировании объектных модулей как сегментов не требуется редактирования связей. Фактически по адресной ссылке $[s, d]$ происходит загрузка требуемого модуля. Сегментация также облегчает выполнение параллельно используемых программ. Для этого адрес программы, занимающей отдельный сегмент, вносится в несколько таблиц сегментов.

Страничная организация памяти. При этом способе управления памятью пространство адресов разбивается на блоки фиксированной длины, называемые *страницами*. Программы и данные делятся на так называемые логические страницы того же размера, что и физические. Адреса при страничной организации образуются аналогично, как и при сегментной адресации. Каждый адрес представляет собой пару $[p.\ d]$, где p – имя страницы: d – смещение относительно начала страницы.

Каждый процесс (задание) имеет таблицу страниц (ТСт), включающую список страниц. Каждая запись таблицы страниц содержит: признак нахождения страницы в памяти, указание на местоположение страницы, бит защиты для контроля способа доступа. Адрес таблицы хранится в регистре таблицы страниц и соответствует выполняемому в данный момент процессу. Если во время выполнения программы встречается адрес $[p.\ d]$, то по содержимому регистра таблицы программа управления памятью находит нужную таблицу страниц, в p-й записи которой находится информация об искомой странице.

Рис. 1.9.
Вычисление адреса при страничной организации.



Если нужная страница расположена в основной памяти, то искомое слово расположено со смещением d от начала страницы (рис. 1.9). Таким образом, для поиска слова происходят два обращения к основной памяти: одно – к таблице страниц, другое – к странице. Для сокращения времени обращения используется аппаратная поддержка для реализации таблицы страниц. Если требуемой страницы нет в основной памяти, вырабатывается сигнал прерываний, по которому супервизор страниц вводит нужную страницу из внешней памяти, осуществляя при необходимости замещение одной из находящихся страниц в основной памяти.

Фрагментация. Этот термин означает наличие свободных участков в памяти. Когда эти участки возникают между логическими блоками, говорят о внешней сегментации (сегментная организация) из-за неравенства сегментов при замещении. При страничной организации имеет место внутренняя фрагментация, поскольку не всегда можно разделить программу на целое число страниц. Принятая длина страниц колеблется от 256 байт до 1 К байт.

Для уменьшения фрагментации используют третий подход, сочетающий сегментную и страничную организации. Он заключается в том, что память разбивается на сегменты, а сегменты – на страницы. Имеются две таблицы – сегментов и страниц внутри сегмента. Адрес слова состоит из трех компонентов [s, d, p]. Вход в таблицу сегментов осуществляется через регистр сегментов, s-я ячейка содержит базовый адрес и длину таблицы страниц для данного сегмента. Компонент адреса p определяет запись в таблице страниц, которая указывает на p-ю страницу. Для нахождения адреса слова надо смещение d прибавить к базовому адресу страницы. Итак, адрес означает d-е слово p-й страницы s-го сегмента. Для поиска слова в этом случае необходимы три обращения к памяти, что даже при аппаратной реализации требует значительного времени. С целью его сокращения можно использовать группу ячеек памяти, которые сопоставляют каждой паре ps физические адреса страниц. Если требуемой комбинации ps нет ни в одной ячейке, происходит прерывание, и супервизор заменяет содержимое одной из ячеек новой ссылкой ps.

Виртуальная память. Это фиктивная память, которая определяется полем адресов пользователей. В системе с виртуальной памятью у пользователя поддерживается иллюзия, что необходимая ему информация находится в основной памяти. Если процесс обращается к информации, которая находится не в основной памяти, вырабатывается сигнал прерывания к супервизору. Супервизор перемещает запрашиваемую информацию в основную память, и процесс получает к ней доступ.

Виртуальная память может базироваться на страничной, сегментной и смешанной стратегии. При реализации обменов между основной и внешней памятью возникают следующие задачи: выбор методов распределения памяти для каждого ее уровня; определение критериев для выбора перемещения страниц во времени; выбор критериев вытеснения страниц в основной памяти. Первая задача сводится к разделению основной памяти на физические страницы и корректированию таблицы занятых и свободных страниц.

При сегментации одним из способов управления памятью является использование списка свободных областей (пустот памяти), упорядоченного по возрастанию их адресов. Каждая область списка содержит указатель следующей области и длину – количество блоков данной области. Выделение памяти можно производить по одной из следующих стратегий: выбор требуемой области и наименьшей требуемой области. В обеих стратегиях отыскивается область необходимого размера, а оставшаяся часть памяти возвращается в список пустот. Вторая стратегия требует просмотра всего списка пустот. Если этот список упорядочить по возрастанию длин областей, то алгоритм поиска первой требуемой области такой же, как и наименьшей требуемой.

Вторая задача по принятию решения о времени передачи страниц из внешней памяти известна как проблема стратегии подкачек. Перемещение страницы возможно по запросу и с опережением. Стратегия подкачек с опережением может дать выигрыш во времени, если запрос на нее последует через некоторое время, и проигрыш – если вызванная в память страница не потребуется. В большинстве случаев стратегия подкачек по запросу выгоднее.

Третья задача связана с вытеснением из оперативной памяти физической страницы при замене ее логической. Имеются различные критерии удаления, но основным всегда остается минимум перемещений, т. е. удаляются страницы, обращение к которым минимально. Рассмотрим наиболее распространенные.

1. Случайный выбор – удаляется случайная страница.

2. Первый пришел – первый ушел (FIFO). Удаляется самая старая страница, которая дольше всех находилась в основной памяти. Стратегия применима как в пределах памяти отдельных процессов, так и во всей памяти. Ее несложно реализовать, но при плотной загрузке системы вытесняются нужные страницы.

3. FIFO с переменным приоритетом. Каждому активному заданию i выделяется $n(i)$ страниц, где $n(i) \neq n(j)$ при $i = j$. В пределах каждой группы из $n(i)$ страниц применяется стратегия FIFO. Через каждые t единиц времени значения $n(i)$ для вмех i меняются.

4. Удаления по давности. Удаляется страница, к которой не обращались дольше всех. Реализация стратегии связана с подсчетом обращений к каждой странице в единицу времени и удалением той, для которой это число минимально.

5. Предписанные приоритеты. Программист или компилятор может назначить приоритеты страницам, а система удаляет страницу на их основании, т.е. по минимальному приоритету.

6. Приоритеты системы. Они устанавливаются на основании приоритетов заданий, удаляется страница с наименьшим приоритетом.

1.6. ФАЙЛОВЫЕ СИСТЕМЫ

Понятия. *Файлом* является физическое представление информации о совокупности объектов. Он состоит из записей, каждая из которых содержит поля. Система управления файлами – это часть ОС, которая реализует работу с файлами. Она обеспечивает следующие функции: эффективное распределение внешней памяти, гибкость доступа к данным, максимальную независимость от аппаратуры, совместное использование общих файлов, безопасность информации, эффективность реализации работы с файлами.

Файлы хранятся обычно на дисках и лентах. Первая задача заключается в эффективном распределении памяти на этих устройствах. Физическое пространство разделяется на блоки фиксированной или переменной длины. В случае страничной организации памяти размер блока зависит от размера страницы. Файл представляет собой совокупность блоков, которые связаны между собой, образуя списки. Стратегия распределения внешней памяти аналогична стратегии распределения основной памяти, обсужденной выше.

Вторая задача заключается в именовании файлов. Пользователи, работая с разделами или файлами, предпочитают иметь свою систему имен. Регистрируя имена файлов, система управления файлами оформляет записи в справочниках. Пользователь обращается к файлу по имени, для которого в справочнике из данного контекста существует запись. По ней определяется фактическое расположение файла. Это может быть осуществлено поиском более чем на одном уровне справочника. Запись справочника может содержать: символическое имя файла, его местонахождение, уникальный идентификатор, одинаковый для всех пользователей файла, тип доступа, дату последнего использования.

Система управления файлами должна выполнять много операций для реализации запроса. Одни из них логические (установка уникального имени по локальному), другие физические (обмен блоков информации). Эти различные функции могут описываться последовательностью уровней программного обеспечения. Данные уровни образуют пять основных подсистем системы управления файлами (рис. 1.10). Каждый из уровней реализует свои возмож-

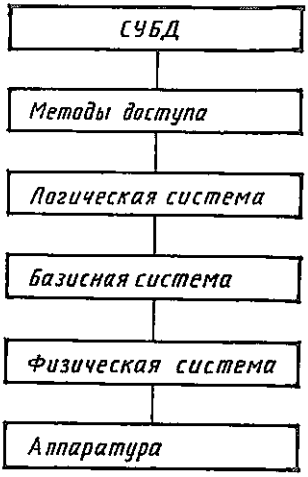


Рис. 1.10.
Подсистемы файловой системы.

ности, используя функции нижележащего уровня. Аппаратура образует нижний уровень.

Система ввода-вывода. Аппаратура ввода-вывода включает каналы (контроллеры) и сами устройства. Команда НАЧАТЬ ввод-вывод задает адрес устройства, канала и косвенно программы канала, которая расположена в памяти и управляет вводом-выводом. Канал выполняет канальную программу, нормальное или аварийное завершение которой вызывают прерывания. Процессор, обрабатывая прерывания, анализирует завершение операции ввода-вывода. Система ввода-вывода рассматривает устройства как взаимодействующие процессы дисковода. На этом уровне выполняются следующие команды: РАЗМЕТИТЬ диск на блоки, ПРОЧИТАТЬ (ЗАПИСАТЬ) метку, ПРОЧИТАТЬ (ЗАПИСАТЬ) блок информации.

Базисная система. Она определяет физические адреса данных путем обработки дескрипторов файлов, которые и дают адреса блоков. Кроме того, при помощи связи с оператором эта система следит за установкой томов. В дескрипторе содержатся адреса каждого блока файла и управляющая информация. Для работы с дескрипторами в состав базисной системы входит ряд команд, среди которых можно выделить следующие: СОЗДАНИЕ дескриптора файла, СОЗДАНИЕ поддескриптора файла (для подмножества файла), ЧТЕНИЕ дескриптора файла (помещение в область основной памяти), ОСВОБОЖДЕНИЕ дескриптора (помещение дескриптора во внешнюю память).

Для управления томами базисная система содержит таблицу всех томов системы, в которой содержится информация о их состоянии. Для управления томами используются следующие возможные команды: ЗАПРОСИТЬ (V), ОСВОБОДИТЬ (V), ПРИКРЕПИТЬ (V), ОТКРЕПИТЬ (V), где V – имя тома. При наличии базисной системы ОС может работать с дескрипторами файлов и выполнять команды более высокого уровня (читать и писать информацию).

Логическая система. Эта система обеспечивает преобразование пользовательских имен файлов в уникальные идентификаторы, а также готовит файл для операций чтения и записи. Информация о файле может быть получена в два этапа. Для поиска соответствующей записи в справочнике пользователя применяется локальное имя файла. Определяется идентификатор файла, по которому в главном справочнике всех файлов находится запись с уникальным именем.

Логическая система реализует команды, которые позволяют пользователю создавать файл, читать из файла и записывать в него. Для извещения логической системы о работе с каким-либо файлом пользователь должен сделать его доступным, выполнив команду открытия файла. По этой команде логическая система определяет положение файла и подготавливает его к выполнению записи или чтения.

Основные команды логической системы могут быть заданы шестеркой S = (С, У, О, З, Ч, П), где С – создать; У – уничтожить; О – открыть; З – закрыть; Ч – читать и П – писать соответственно файлы. В команде создания указывается имя файла, его размер и том размещения, при уничтожении, открытии и закрытии – только имя файла.

Команда ЧИТАТЬ (имя, N, M, K) выдает запрос на чтение в файле заданного количества блоков K, начиная с блока N, которые помещаются в основную память по адресу M. Команда ПИСАТЬ (имя, N, M, K) делает запрос на

запись заданного количества блоков K в файл с указанным именем. Запись начинается с блока N из основной памяти по адресу M. При выполнении каждой из этих команд возможны ошибочные ситуации. Например, при создании файла он может быть не создан из-за повтора имени, отсутствия тома и других причин, для которых в системе должны быть логические средства обработки.

Методы доступа. Пользователям часто необходимо иметь доступ к определенным записям в файле, которые задаются полями записи. Существует определенный механизм в системе, выполняющий эти функции с помощью методов доступа. Имеется несколько способов определения записей в файле. Простейший состоит в использовании положения записи, при этом поиск ведется по номеру записи. Более сложные предусматривают задействование некоторых полей данных в качестве ключей.

Для задания требований на поиск информации пользователь выдает запрос. Если запрос задан в терминах ключей, использованных при организации данных, система производит быстрый поиск. Известны разные подходы к его реализации. В одном из них производится последовательный поиск записей, пока не будут найдены все, содержащие заданные значения ключей. Этот способ прост для программирования, но требует значительного времени для исполнения. В другом варианте для каждого значения ключа заводится указатель на все записи с этим ключом. Множество таких указателей образует инвертированный файл (таблицу индексов). Поиск при этом ускоряется, но требуется создание дополнительных файлов. При третьем подходе организуется цепочка всех записей, которые имеют одинаковое значение частичного ключа. Поиск ускоряется, но на указатели расходуется много памяти, и их необходимо обновлять при изменении ключевых полей в записи.

ОС обычно предоставляют стандартизованные методы доступа, которые позволяют работать с записями, используя стандартный набор команд. Методы доступа ОС имеют две характеристики. Они могут быть базисными или с очередями, а также последовательными или прямыми. При *базисном методе доступа* пользователь должен предусмотреть буферизацию записей. При *доступе с очередями* ОС имеет стандартную систему буферизации, пользователь должен только выполнять соглашения, связанные с буферизацией. Последовательный метод доступа организует записи файла в последовательном порядке, прямой метод – в произвольном. Прямой метод не исключает последовательного, который может быть модифицирован с использованием индексов для поиска.

Система управления базой данных (СУБД). Обычно в прикладных программах принимаются различные допущения о структуре и форматах файла, что в принципе требует нескольких его вариантов. Кроме того, требуется модифицировать программы при изменении форматов файлов. Для достижения независимости форматов файлов и программ пользователя, а также доступа к одному файлу из различных программ используется определенным образом организованная совокупность данных – база данных и ее система управления. Последняя является прикладной системой, налагающей несколько логических структур на данные в соответствии с потребностями программ. Для этого СУБД может использовать инвертированные файлы, индексы, ключи и т. д. Применяются также языки описания и запросов к данным.

Выводы

1. Пользователь, программист и системный программист могут работать с различными виртуальными машинами, включающими свой набор языковых средств для организации вычислительных процессов на каждом уровне иерархической структуры ВС.

2. С функционированием ОС связано несколько ключевых задач организации вычислительных процессов на логическом уровне: эффективное преобразование виртуальных машин, распределение ресурсов, взаимодействие процессов и поддержание пользовательского интерфейса.

3. Для управления процессами на нижнем уровне используются блокировка памяти и механизм семафоров, а также сигналы синхронизации. На верхнем уровне для взаимного исключения доступа к разделяемым ресурсам применяются мониторы, для синхронизации – почтовые ящики (рандеву).

4. Для назначения процессов на процессоры используются различные стратегии диспетчеризации: выделением постоянного или переменного кванта времени, в порядке поступления, в обратном порядке, по статическому или динамическому приоритету и др.

5. Память распределяется статически или динамически между процессами. Для более эффективного ее использования применяются сегментная и страничная организации. Для работы пользователей в адресном пространстве, превышающем физический объем основной памяти, реализуется виртуальная память. Расширение основной памяти происходит за счет обмена страниц (сегментов) между основной и внешней памятью.

6. Для работы с внешней памятью организуется файловая система. Иерархия последней включает (снизу вверх) управление канальными программами (драйверами), работу с дескрипторами и томами, работу с файлами, организацию различных методов доступа к данным. Наконец, как расширение файловой системы реализуется база данных.

Рекомендуемая литература: [2, 8, 26, 28].

Вопросы для контроля

1. Определите состав виртуальных ФОРТРАН- и ПЛ-машин.
2. В чем отличие систем с разделением времени и реального времени?
3. В чем заключаются функции управления процессами, ресурсами, вводом-выводом?
4. В чем достоинства и недостатки семафоров, мониторов?
5. Охарактеризуйте методы управления процессором.
6. В чем отличие страничной и сегментной организации памяти?
7. Сравните методы вытеснения страниц при виртуальной памяти.
8. Покажите изменение функций в подсистемах файловой системы.

2. ОРГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ В ЕС ЭВМ

2.1. СОСТАВ И ФУНКЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ЕС ЭВМ

Структура программного обеспечения. Система ПО ЕС ЭВМ представляет собой набор основных операционных систем, комплексов программ технического обслуживания (КПТО), пакетов прикладных программ (ППП). Операционные системы включают ДОС ЕС, ОС ЕС и СВМ ЕС. КПТО содержат тест-программы, работающие без ОС и под ее управлением. По функциональному назначению тесты подразделяются на наладочные, проверочные и диагностические. Наладочные служат для проверки функционирования узлов и блоков машины во время наладки, проверочные – во время эксплуатации, диагностические – для обнаружения и локализации неисправностей.

Организация ОС ЕС представляет собой совокупность программ, таблиц и указателей, предназначенных для обеспечения режимов мультипрограммирования. Часть модулей хранится в основной памяти, другие помещаются во внешней и считываются в основную по мере необходимости. Логически ОС ЕС состоит из управляющей и обрабатывающей программ.

Управляющая программа (УП) предназначена для организации вычислений в системе и управления аппаратными средствами. Она организует обработку потока заданий, управление процессами и ресурсами и работу системы в целом. *Обрабатывающая программа* включает средства программирования (трансляторы, редакторы и т. д.).

Задание – основная независимая единица работы, состоящая из одного или нескольких пунктов. Выполнение пункта задания – это выполнение системной или прикладной программы. Основные функции УП – управление заданиями, задачами, данными, восстановлением ВС и диагностика. Перед началом работы ОС для загрузки ядра в оперативную память используется программа первоначальной загрузки и инициализации ядра.

Программа первоначальной загрузки осуществляет настройку оперативной памяти, загрузку ядра ОС и программы его инициализации. Последняя подготавливает системные таблицы, настраивает системные наборы данных, проверяет состояние внешних устройств, связывается с оператором для оперативного изменения свойств ОС.

Управление заданиями. В этот компонент входит главный планировщик и планировщик. Главный планировщик осуществляет связь между оператором ЭВМ (командами) и ОС (сообщениями). Основные функции планировщика: ввод потоков заданий и запись их в системные очереди, инициирование выполнения шагов заданий, вывод результатов заданий. Как только начинается выполнение шага задания, управление заданиями прерывается и осуществляется модулями управления задачами. Планировщик включает три компоненты: программу системного ввода (ридер), программу инициирования шагов заданий (инициатор) и программу системного вывода (райтер).

Управление задачами. Функции модулей этой компоненты состоят в обеспечении потребностей программы в процессе ее выполнения. Функции управления задачами следующие: распределение основной памяти, загрузка программ, планирование выполнения программ, переключение управления от одной задачи к другой, защита задач от непредусмотренного взаимного влияния. Основой программ управления задачами является супервизор, главная часть которого в виде ядра все время находится в оперативной памяти.

Управление данными. Этот компонент обеспечивает ввод и вывод данных с последовательной, индексно-последовательной, библиотечной и прямой организациями. Набор данных – это совокупность логически связанных записей, имеющих определенную организацию. Характеристики наборов данных описывают организацию данных, их местоположение, объем и записываются во внешней памяти вместе с набором данных.

Программное обеспечение *телеобработки* позволяет осуществлять передачу информации в ВС и из нее по каналам связи (телефонным, телеграфным и т. д.). Средства телеобработки обеспечивают одновременную работу с ВС нескольких удаленных абонентов, для чего используются телекоммуникационные методы доступа.

Программные средства машинной графики позволяют работать ВС с алфавитно-цифровыми и графическими дисплеями, включают программы графического метода доступа, средства разработки программ графических приказов, пакеты графических подпрограмм. В результате трансляции исходных модулей получаются объектные.

Объединение нескольких объектных модулей в один загрузочный модуль (ЗМ) и подключение необходимых подпрограмм выполняет редактор связей. Загрузочный модуль, включающий машинные команды, информацию о внешних ссылках, резервируемой памяти, настройке адресов при загрузке в память, помещается в библиотеку загрузочных модулей. Вызов загрузочного модуля в память и настройка его адресов осуществляются программой выборки. Загрузчик выполняет отдельные функции редактора связей по созданию загрузочного модуля, помещает его, минуя библиотеку загрузочных модулей, в память и передает ему управление. Загрузчик в основном используется для однократного создания и выполнения загрузочного модуля.

В зависимости от структуры загрузочный модуль в ОС ЕС может загружаться в память полностью или частично. Допускаются следующие структуры загрузочных модулей: простая, оверлейная, динамические последовательная и параллельная.

Модуль *простой структуры* загружается в основную память как единое целое. В модуле *оверлейной структуры* указываются сегменты, которые перекрывают друг друга в основной памяти.

Динамические последовательные структуры используются при большом количестве сегментов и их вызове в зависимости от обрабатываемых данных. Вызов сегментов в память производится макрокомандами супервизора по мере необходимых условий. В *динамических параллельных структурах* модули могут выполняться в режиме мультипрограммирования.

В ОС ЕС допускается повторное использование модулей. В зависимости от этого различают однократно используемые, повторно используемые и повторно входимые (реентерабельные модули). Тип модуля определяется во время

редактирования связей программистом.

Однократно используемые модули модифицируются в процессе выполнения, для повторного их использования необходима перезагрузка. *Повторно используемый модуль* выполняется многократно, при этом запросы на его выполнение удовлетворяются в порядке очереди.

Реентерабельные модули могут поочередно не до конца использоваться несколькими программами. В момент выполнения такого модуля одной программой и при ее прерывании к нему может обратиться другая программа. При этом динамически выделяется память для рабочей области и области сохранения каждой вызывающей программы.

Система отладки предназначена для обнаружения логических ошибок в программах. Исходной информацией является отлаживаемая программа, включающая объектные модули, из которых формируется секция с запросами программиста.

Рассмотрим функционирование ОС ЕС при выполнении простого задания по трансляции, редактированию связей и выполнению программы. Исходная колода карт считывается модулем управления заданием (ридером) и помещается в библиотеку входных очередей. Через некоторое время, когда освобождается раздел оперативной памяти, другим модулем управления заданиями (инициатором) считывается первая карта EXEC потока управления заданиями и выполняется загрузка и запуск компилятора. Если в процессе работы компилятора необходима помощь ОС (выполнение ввода-вывода), она обеспечивается модулем управления задачами. Компилятор завершает работу записью объектного модуля в соответствующую библиотеку (с помощью управления задачами), после чего активизируется модуль управления заданиями (терминатор), который стирает следы отработавшей в данном разделе программы. Теперь инициатор (управление заданиями) загружает из системной библиотеки редактор связей, который считывает объектный модуль, формирует загрузочный модуль и записывает его в библиотеку загрузочных модулей (с помощью программ управления задачами). Инициатор-терминатор (управление заданиями) снова стирает следы отработавшей программы, считывает в память загрузочный модуль и инициирует его выполнение. После этого работают модули управления задачами (обработка прерываний, ввода-вывода, планирование и т. д.). По завершении выполнения программой системного вывода (управление заданиями) результаты помещаются в выходную очередь. При работе с системными и пользовательскими библиотеками подключаются программы методов доступа (управление данными).

2.2. ЯЗЫК УПРАВЛЕНИЯ ЗАДАНИЯМИ

Задания. Для управления прохождением программ в ОС ЕС составляется задание, в котором указываются имена программ и используемые ресурсы. Задание состоит из управляющих операторов языка управления заданиями (ЯУЗ), обеспечивающих связь между пользователем и системой. ЯУЗ включает девять типов управляющих операторов (табл. 2.1).

Основные карты ЯУЗ: JOB – разделение и идентификация заданий; EXEC – идентификация исполняемого модуля или программы; DD – описание входных и выходных наборов данных в задании. Для описания процедур

Таблица 2.1

Тип оператора	Формат
Задания	// имя JOB [операнды] [комментарии]
Пункт задания	// имя EXEC [операнды] [комментарии]
Описание данных	// имя DD операнды [комментарии]
Ограничения	/* [комментарии]
Пустой	//
Комментарии	//* комментарии
Процедуры	// имя PROG [операнды] [комментарии]
Конец процедуры	// имя PEND [комментарии]
Команды	// команда операнды [комментарии]

ЯУЗ во входном потоке используются операторы PROC – начало и PEND – конец процедуры. Оператор команды предназначен для ввода во входном потоке команд оператора управления системой.

Оператор комментария (//*) содержит пояснительный текст, который может выдаваться на печать. Конец исходных данных отмечается оператором ограничения (/*). Пустой оператор (//) означает конец задания.

Все управляющие операторы записываются на бланках в строках по 80 позиций, идентифицируются символами // в 1-й и 2-й колонках. Они могут содержать четыре поля: имя, оператор, операнды, комментарии, которые разделяются одним или более пробелами. Имя идентифицирует оператор, включает не более восьми алфавитно-цифровых символов, начинается с буквы. Поле операции содержит код оператора и определяет тип управляющего оператора.

Поле операторов включает позиционные и ключевые параметры, которые могут быть опущены. *Позиционные* записываются в определенном порядке, отсутствие такого параметра указывается запятой. *Ключевые* параметры записываются в любом порядке, отсутствие параметра не указывается запятой, параметр обозначается ключевым словом, за которым следуют значения.

Несколько заданий образуют пакет во входном потоке. Первый оператор

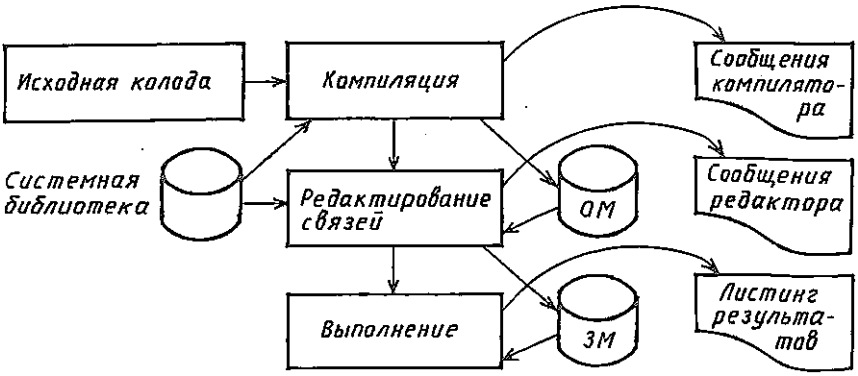


Рис. 2.1.
Прохождение задания при компиляции, редактировании и выполнении.

задания JOB. Задание можно разделить на шаги задания, которые обрабатываются последовательно.

Рассмотрим схему задания, приведенную на рис. 2.1, которая состоит из трех шагов: компиляции, редактирования связей и выполнения. Для программиста это одно задание, но система должна выполнить три программы. Напишем наше задание на условном ЯУЗ без указания параметров, используя лишь известные операторы. При этом для каждого устройства ввода-вывода необходимо описание DD. Последовательность будет следующая:

```
// имя JOB
// имя EXEC шага компиляции
// имя DD исходной колоды на перфокартах
// имя DD описания объектного модуля на диске
// имя DD описания выходных данных печатающего устройства
// имя DD описания библиотеки компилятора на диске
// имя EXEC шага редактирования связей
// имя DD входного объектного модуля с диска
// имя DD описания библиотеки редактора связей на диске
// имя DD описания выходных данных печати сообщений PC
// имя DD описания загрузочного (выходного модуля на диске)
// имя EXEC выполнения загрузочного модуля
// имя DD описания загрузочного модуля (входного)
// имя DD описания листинга результата печатающего устройства
```

Оператор задания. Содержит имя задания, JOB , два позиционных и ключевые параметры, которые необязательны. Позиционные параметры включают учетную информацию (номер отдела программирования, учетный номер программиста, шифр темы и т. д.) и заключаются в скобки. Имя программиста записывается после учетной информации, не может включать более 20 символов и, если содержит символы, отличные от латинского алфавита, заключается в апострофы. Пример:

```
//EXAMPL JOB (25, 73, 40), 'ВИШНЯКОВ В.А.'
```

Для указания класса задания используется параметр CLASS, формат CLASS=n, где n = A...0. По умолчанию принимается класс A. Приоритет задания в очереди данного класса определяется параметром PRTY с форматом PRTY = m, где m = 0...13.

Для указания уровня распечатки сообщений системы используется параметр MSGLEVEL = (n1, n2), n1 = 0, 1, 2 – указывает на печать: 0 – только оператора JOB, 1 – операторов задания и каталогизированных процедур, 2 – только входных управляющих операторов. Подпараметр n2 имеет два значения; 0 – сообщения о распределении устройств не выводятся (при аварийном завершении выводятся); 1 – сообщения о распределении выводятся.

Ряд параметров является общим для операторов JOB и EXEC. К ним относится параметр COND , который указывает условие обхода пункта задания и имеет вид для оператора JOB COND =((число, отношение) ...) . Отношение принимает значения: GT (больше), GE (больше, равно), EQ (равно), LT (меньше), LE (меньше, равно), NE (не равно).

Код возврата определяется числом из диапазона 0–4095, которое для системных обрабатывающих программ кратно 4 и равно нулю при отсутствии

ошибок. Любая программа после завершения вырабатывает код возврата. Если оператор JOB имеет параметр COND = (16, LT), то задание будет выполняться, пока код возврата меньше 16.

В операторе EXEC параметр имеет формат COND =((число, отношение, имя пункта),..), где имя пункта обходят при выполнении условия. Например, если COND = (20, GT, STEP), то данный пункт задания выполняется, если на шаге STEP код возврата больше 20.

Параметр TIME задает максимальную величину времени, в течение которого задание или пункт задания могут использовать процессор. Если время выполнения превышает указанное, задание снимается. Формат параметра TIME = = (мин, с). Число минут не более 1440, секунд – 60. Если указывается первый параметр, скобки опускаются.

Параметр REGION=nK указывает размер раздела памяти, выделяемого для задания или пункта задач в режиме MVT (по умолчанию в стандартной процедуре запуска 50К).

Пример управляющего оператора JOB:

```
//EXAMPL2  JOB (20,25),PETROV, MSGLEVEL=(1, 1), REGION=100K,
//              TIME=20, PRTY=10, CLASS=C, COND=(8, LT)
```

Здесь указываются учетная информация, фамилия программиста, задается распечатка всех операторов задания, включая каталогизированные процедуры, и распечатка устройств ввода-вывода для наборов данных. Выделяется 100К памяти, время выполнения не более 20 мин. Приоритет задания 10 в системной входной очереди С. Пункты задания будут выполняться, если код возврата предыдущих не более 8.

Оператор пункта задания. Основное назначение оператора – указание выполняемой программы или процедуры. В поле имени задается имя, на которое могут ссылаться другие пункты задания.

Имя выполняемой программы задается позиционным параметром PGM= имя программы (имя пункта, имя DD), где имя DD описывает временную библиотеку в пункте задания "имя пункта". Вместо вызова программы в операторе EXEC может быть указан вызов процедуры в системной библиотеке SYS1.PROCLIB или во входном потоке. Тогда первый параметр будет записан PROC=имя процедуры (для каталогизированной процедуры PROC можно опустить).

Для передачи управляющей информации применяется параметр PARM= 'параметры', причем формат параметров зависит от программы.

Для указания приоритета задачи, образованной пунктом задачи, используется параметр DPRTY=(n1, n2), причем текущий приоритет вычисляется в виде n1*16+n2.

Пример оператора EXEC:

```
//STEP1  EXEC PGM=PROG1, PARM=(11,25), COND=(8, LT), TIME=2,
//              REGION=100K, DPRTY=(2, 1)
```

Здесь шаг задания с именем STEP1 связан с выполнением программы PROG1, которой передаются значения 11 и 25. Пункт задания не будет выполнен, если один из предыдущих пунктов задания вырабатывает код более 8.

Для программы будет выделено 100 К байт памяти, время выполнения не более 2 мин, диспетчерский приоритет – 33 (2x16+1=33).

Оператор описания данных. Предназначен для описания наборов данных, используемых обрабатывающей программой, с помощью оператора DD. В нем могут быть указаны имя набора данных, его местонахождение, характеристики, режим обработки и другие параметры. Задавая информацию о наборе данных в различных источниках (макрокоманда DCB, оператор DD, метка набора данных), можно добиться независимости программы от местоположения набора и характеристик.

Каждый набор DD должен иметь имя. Имена наборов данных в программе должны соответствовать именам в операторе DD. Ссылки на оператор DD задаются в одной из следующих форм: *.имя=DD, *.имя пункта.имя DD.

В первом случае имя используется для ссылок на оператор DD в том же пункте, во втором – в другом пункте.

Для ввода данных во входном потоке необходимо перед ним поместить оператор //имя DD*, а после него – /*. Если набор данных во входном потоке содержит управляющие операторы, то ему должен предшествовать DD DATA.

Определенные характеристики набора данных можно получить из другого набора, ссылаясь на него в операторе DD в параметре DD NAME=имяDD.

Само имя набора данных указывается в параметре DSN=имя набора данных, которое может быть простым или составным. Простое начинается с буквы и включает не более 8 символов, составное состоит из простых имен, разделенных точками (не более 44 символов).

Для задания имени набора данных, указанных в предыдущем операторе DD или предыдущем пункте, необходимо соответственно сослаться: DSN=*.имя DD и DSN=*.имя пункта.имя DD. Имя временного набора данных, существующего в рамках задания, определяется DSN=&имя. Набор данных, создаваемый в пункте задания, считается новым, а существовавший ранее – старым (OLD), может быть модифицированным (MOD), разделенным (SHR). После выполнения пункта задания набор может быть сохранен (KEEP), удален (DELETE) либо передан другому пункту задания (PASS), каталогизирован (CATLG), выведен из каталога (UNCATLG). Состояние набора данных до выполнения, после нормального или аварийного завершения пункта задания описывается параметром

$$\text{DISP}=\left(\begin{bmatrix}\text{OLD}\\ \text{NEW}\\ \text{MOD}\\ \text{SHR}\end{bmatrix}\ \begin{bmatrix}\text{,DELETE}\\ \text{,KEEP}\\ \text{,PASS}\\ \text{,CATLG}\\ \text{,UNCATLG}\end{bmatrix}\ \begin{bmatrix}\text{,DELETE}\\ \text{,KEEP}\\ \text{,CATLG}\\ \text{,UNCATLG}\end{bmatrix}\right)$$

Все подпараметры параметра DISP необязательные. В отсутствии третьего подпараметра при аварийном окончании пункта задания состояние набора данных определяется вторым подпараметром.

Устройство, на котором располагается набор данных, описывается параметром UNIT=устройство, где устройство – физический адрес, типовое или групповое имя (SYSDA, SYSSQ). Например, UNIT=280 (5010, TAPE, SYSDA) задает набор на магнитной ленте, заданной адресом 280, именем 5010 или групповым именем TAPE, SYSDA.

Для указания объема памяти, выделяемого набору данных, используется параметр SPACE в виде

$$\text{SPACE}=\left(\left\{\begin{array}{l}\text{TRK}\\ \text{CYL}\\ \text{дл. блока}\end{array}\right\},(\text{n1}\ [,\text{n2}[,\text{оглавление}]]),\text{RLSE}\right),$$

где в фигурных скобках указывается единица измерения памяти (TRK – в дорожках, CYL – в цилиндрах, дл. блока – размер блока в байтах); n1 – количество единиц памяти перед выполнением пункта задания, n2 – объем памяти, выделенный дополнительно; оглавление имеет смысл для библиотечных наборов, выделяется для оглавления в блоках по 256 байт. Подпараметр RLSE указывает, что после создания набора данных вся выделенная, но неиспользованная память освобождается.

Параметр VOLUME (VOL) задает информацию о томе, на котором размещается набор данных. Каждый том имеет регистрационный номер, при специальном запросе он указывается. В первом случае формат параметра VOL=SER= (список регистрационных номеров).

Параметр SYSOUT=X записывают, когда требуется направить набор данных в выходную очередь, где X = A –Z, X = 0–9. Набор данных, используемых в прикладной программе, может быть описан макрокомандой DCB. Некоторые недостающие сведения, такие, как характеристики записей набора данных и режимы его обработки, можно указать в параметре DCB=(список), где "список" – список ключевых подпараметров. Эти подпараметры совпадают с параметрами макрокоманды DCB. Более подробно с ЯУЗ можно познакомиться в книге [8].

Рассмотрим пример записи оператора DD:

```
//M1 DD  DSN=X,  DISP=(,PASS),UNIT=5061,
         VOL=SER=111111, SPACE=(240,50)
```

Оператор M1 определяет набор данных X, который передается следующим пунктам задания (указано в параметре DISP). Этот набор размещен на томе 111111, который монтируется на одно из устройств ЕС-5061. Набор требует для размещения 50 записей (блоков) по 240 байт.

Обработка заданий. Рассмотрим процесс обработки заданий и основную информацию, формируемую при этом. Программа системного ввода получает от главного планировщика информацию об устройстве ввода. Запускается несколько программ системного ввода, которые конкурируют из-за времени центрального процессора. Программа системного ввода выполняет следующие функции: считывает входной поток заданий, просматривает и контролирует операторы ЯУЗ и фомирует управляющие таблицы, копирует данные из входного потока на внешнюю память и создает блок управления данными, строит запись из таблиц, созданных для каждого задания, интерпретирует управляющие операторы задания. Для первого JOB из потока заданий строится таблица.

При распознавании оператора EXEC выясняется, вызывается программа или процедура. Для программы создается таблица управления шагом, в которую заносится информация из параметров оператора EXEC. При вызове процедуры разыскивается процедура либо в начале задания, либо в библиотеке каталогизированных процедур.

При обработке оператора DD строятся таблица параметров файла (ТПФ) и блок управления файлами задания (БУФ). БУФ содержит информацию, относящуюся к устройству ввода-вывода и состоянию файла данных. ТПФ создается для описания структуры файла, его записей.

Для каждого задания строится блок системных сообщений, в который будут заноситься все сообщения управления заданиями.

Процесс ввода заданий выполняется в режиме интерпретации, т. е. ввода и обработки каждого оператора. После завершения обработки операторов программа системного ввода преобразует построенные таблицы и подает на вход очереди заданий.

После завершения системного ввода вызывается планировщик, который выбирает очередную запись из очереди заданий, подготавливает информацию для создания шага задания.

2.3. НАЗНАЧЕНИЕ И СТРУКТУРА СУПЕРВИЗОРА

Состав. Супервизор объединяет группу управляющих программ, осуществляющих управление задачами, памятью, временем, вводом-выводом, прерываниями. Он получает управление при возникновении прерываний, обработка и анализ которых являются его основной функцией. Кроме того, супервизор инициирует операции ввода-вывода, распознает ошибки устройств ввода-вывода, обеспечивает ведение системного журнала, управляет загрузкой транзитных программ и содержит программы методов доступа к системным файлам.

Супервизор имеет модульную структуру, хранится в двух системных библиотеках: ядра SYS1.NUCLEUS и транзитов SYS1.SVCLIB. Во время функционирования в памяти постоянно находится ядро супервизора. Состав ядра переменный, задается при генерации системы.

Назначение супервизора. При возникновении любого прерывания процессор переключается в состояние анализа прерывания, кроме прерываний от схем контроля. В этом состоянии выполняется распознавание прерываний и передача управления соответствующей программе обработки прерываний. С точки зрения процессора супервизор представляет совокупность программ анализа и обработки прерываний.

Прерывание по супервизору возникает в случае выполнения программы, содержащей команду обращения к супервизору SVC. Эти команды используются системными и обрабатывающими программами при выполнении операций ввода-вывода, загрузки модуля программы в память, открытия или закрытия файла, завершении программы. В общем случае SVC-команда используется для запроса функции супервизора. Указание, какая из функций требуется, содержится в самой команде SVC в качестве ее параметров. Прерывание по супервизору не маскируется. Программы ОС содержат SVC-команды в явном виде, проблемные используют набор макрокоманд, которые запрашивают выполнение функций супервизора.

Программные прерывания возникают при появлении ошибок во время выполнения программы (деление на нуль, переполнение, исчезновение порядка, неправильная адресация, неправильная операция и т. д.). При появлении такого прерывания супервизор либо передает управление самой проблемной

программе, либо прекращает ее выполнение. Управление будет передано проблемной программе, если до появления сбоя супервизору было сообщено о наличии подпрограммы обработки сбоев специальной макрокомандой. Такой макрокомандой в ДОС ЕС является STXTT, в ОС ЕС – SPIE. Если супервизор не получил сообщения этой макрокоманды, то при появлении программных прерываний он снимает задание, для которого выполнялась программа. При этом может быть распечатка дампа памяти и содержимого регистров.

Прерывания от ввода вывода дают возможность синхронизировать работу центрального процессора и периферийных устройств. Запросы, поступившие одновременно, хранятся в канале до тех пор, пока не будут обработаны в соответствии с приоритетами. В результате прерывания ввода-вывода канал предоставляет супервизору информацию о завершении передачи данных, а также о необычных условиях, возникающих в каком-либо интерфейсе.

Прерывание от ввода-вывода может произойти только при незамаскированном канале, т. е. если бит маски (биты 0–6) в текущем слове состояния программы (ССП) установлен в "1".

Причинами внешних прерываний являются запросы, поступившие по линии связи: сигналы от таймера и кнопки "Прерывание" на пульте управления. Супервизор передает управление модулю времени (прерывание от таймера) либо главному планировщику. Это пребывание тоже маскируется (бит 7 текущего ССП).

Расположение супервизора. Физически супервизор вызывается в область управляющей программы, расположенной в начальной области памяти. Основная память (ОП) при работе супервизора делится на фиксированную и динамическую области (рис. 2.2). В фиксированной области находятся системные управляющие программы. В динамической области располагаются программы пользователя, данные, управляющие блоки и таблицы, программы системного ввода и вывода.

Фиксированная область включает ядро, транзитные области SVC-программ *2, 3* и супервизора ввода-вывода *1* , а также область системных очередей. По способам передачи управления и выполнения SVC-программы бывают четырех типов. SVC-программы первого типа резидентные, не допускающие прерывания (GETMAIN, FREEMAIN, EXIT, EXCP, WAIT). Ко второму типу относятся резидентные программы, допускающие прерывания (LINK, XCTL, LOAD), как правило, по вводу-выводу. SVC-программы первого и второго типов входят в состав ядра супервизора.

SVC-программы третьего типа являются транзитными, допускают прерывания, их объем не превышает 1 К байта (WTO, LOCATE). К четвертому типу относятся транзитные программы, допускающие прерывания и имеющие объем более 1 К байта памяти (OPEN, CLOSE). Они загружаются из библиотеки транзитов SYS1, SVCLIB по частям. В транзитную область ядра загружаются SVC-программы третьего и четвертого типов, в транзитную область ввода-вывода – программы супервизора по обработке ошибок.

За ядром располагается область системных очередей, в которых хранятся данные о задачах, функционирующих в системе, строятся блоки управления задачами (англ. TCB), организуются очереди для системных ресурсов, выделяются буферы для задачи главного планировщика и т. д.

Для режима MFT перечисленные области располагаются за динамической

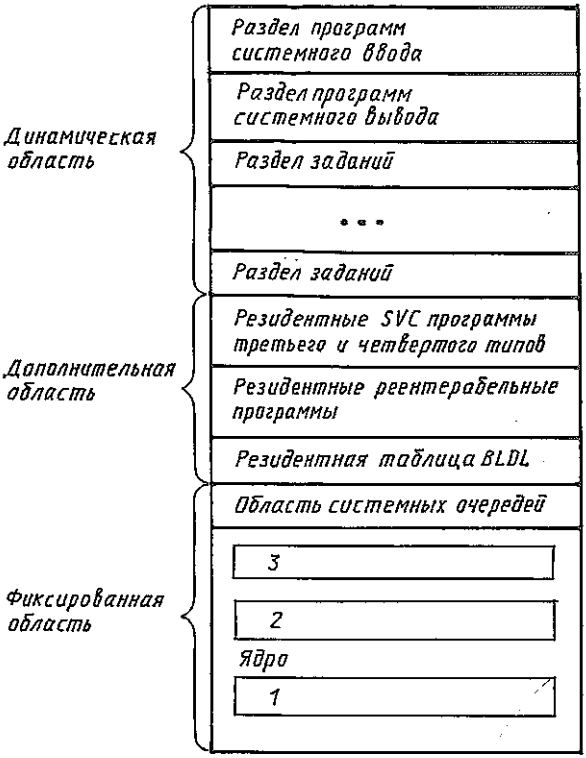


Рис. 2.2.
Расположение программ супервизора в ОП (режим MFT).

областью. Динамическая область в режиме MFT разбивается на фиксированное число разделов, количество которых задается при генерации ОС. В режиме MVT разделы в динамической области не фиксированы, а образуются в момент подготовки пункта задания.

Функции супервизора. С точки зрения выполняемых функций супервизор можно разбить на две группы: управление задачами и вводом-выводом. Программы управления задачами предназначены для анализа прерываний, обработки прерываний, управления распределением процессора, управления основной памятью и программных ресурсов. Поэтому супервизор включает: супервизоры прерываний, задач, оверлейных структур, памяти, времени, страниц.

Супервизор прерываний анализирует поступившее прерывание и определяет путь его обработки. *Супервизор задач* хранит информацию о всех задачах, имеющихся в системе, устанавливает порядок использования требуемых программ для выполнения задач.

Супервизор оверлейных структур дает возможность загрузки транзитов в память, хранит о них сведения, определяет необходимость их нахождения в памяти по мере выполнения. *Супервизор памяти* выделяет и освобождает основную память, хранит информацию о всех ее используемых областях. *Супервизор времени* устанавливает и обрабатывает временные интервалы на основе

информации, содержащейся в макрокомандах таймера. *Супервизор страниц* осуществляет стратегию обмена между основной и внешней памятью в режиме виртуальной памяти.

Макрокоманды супервизора. Для активизации программ супервизора используются макрокоманды, макрорасширение которых содержат SVC-команды. Программы, соответствующие SVC-командам, реализуют функции супервизора. Часть SVC-команд является обязательной, другие могут включаться в процессе генерации системы.

По функциональному назначению макрокоманды делятся на макрокоманды управления связями, формирования задач, управления запросами на основную память, управления ресурсами, работы с таймером.

Программы управления связями при выполнении модулей простой и оверлейной структур ищут и загружают в память необходимые сегменты (CALL). В программах с динамической структурой для вызова загрузочных модулей из библиотеки и работы с ними используется ряд макрокоманд супервизора (LINK, LOAD, DELETE, XCTL, IDENTIFY и др.).

Программы формирования задач позволяют образовывать и уничтожать подзадачи (ATTACH, DETACH), синхронизировать выполнение отдельных сегментов (WAIT, POST), получать информацию управляющих полей задачи (EXTRACT).

Программы управления запросами на основную память включают выдачу и освобождение памяти (GETMAIN, FREEMAIN).

Программы управления ресурсами используют программы запроса на ресурс (ENQ) и его освобождение (DEQ).

Модульность супервизора позволяет настраивать его на конкретную конфигурацию ВС, при этом требуемые функции супервизора задаются при генерации ОС. С помо чью набора макрокоманд генерации описываются необходимые свойства супервизора. После трансляции этих макрокоманд ассемблером получается вариант программ супервизора, который редактируется, каталогизируется и помещается в библиотеках системы.

2.4. УПРАВЛЕНИЕ ПАМЯТЬЮ

Общие положения. В режимах с фиксированным (MFT) и переменным (MVT) числом задач программы пользователя находятся в динамической области основной памяти. Эта область в режиме MFT делится на фиксированное число разделов, количество и размеры которых устанавливаются при генерации. Перераспределение памяти между разделами осуществляется оператором. В разделах могут выполняться проблемные задания и системные программы управления заданиями: ввода, вывода и инициирования. Для заданий может быть выделено не более 15 разделов, для программ системного ввода – не более 3, для программ системного вывода – не более 36. Инициатор выполняется в разделах заданий в промежутках между завершением предыдущего пункта задания и началом следующего.

Программы системного ввода-вывода могут быть как резидентными, так и транзитными (определяется при генерации ОС).

В режиме MVT динамическая область распределяется автоматически между проблемными заданиями и системными программами в виде зон. Зоны вы-

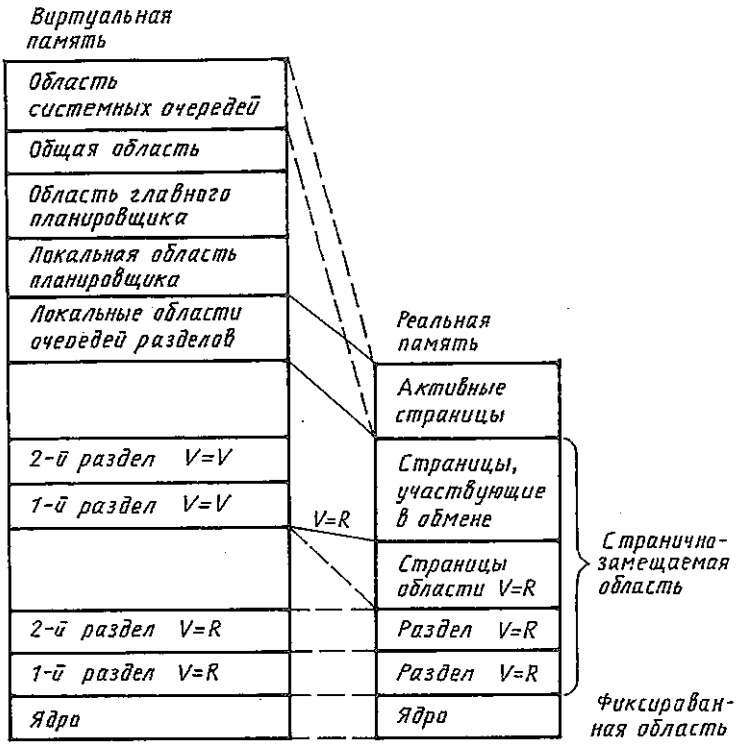


Рис. 2.3.
Организация ОП в режиме виртуальной памяти (режим SVS).

деляются в соответствии с запросами программ и остаются, пока эти программы существуют. В режиме свертки заданий имеется возможность получения дополнительной памяти для задания.

Режим мультипрограммирования с переменным числом задач и виртуальной памятью (SVS) может быть организован в моделях ЕС ЭВМ-2. Вся виртуальная память делится на две области: фиксированную и динамическую (рис. 2.3). В *фиксированной области* размещаются: ядро, область системных очередей, общая область и область главного планировщика. *Динамическая область* используется для выполнения заданий пользователя и системных программ управления заданиями.

В режиме SVS создаются разделы двух типов с виртуальными адресами V-V (виртуальный – виртуальный) и V-R (виртуальный – реальный). Разделы V-V и V-R по-разному отображаются на реальную память (см. рис. 2.3). Первые попадают в странично-замещаемую часть ОП, вторые – в странично-незамещаемую.

Виртуальные страницы, попавшие в странично-незамещаемую область, находятся в ней постоянно, а попавшие в странично-замещаемую могут вытесняться из нее в набор данных SYS1.PAGE и вновь загружаться, не обязательно в тот же раздел ОП, но в странично-замещаемую часть.

Память раздела. Каждое задание выполняется в разделе, который используется всеми его программами. В процессе его выполнения память раздела выделяется по запросам и освобождается по мере завершения задания. Поэтому возникает необходимость динамического распределения памяти для минимизации ее использования. Однако это может привести к фрагментации, т. е. образованию "дыр".

Распределение памяти внутри раздела пункта задания следующее. Сначала располагается программа, вызываемая по EXEC, затем следуют программы, вызываемые по макрокомандам LINK, XCTL, ATTACH, для которых организуется область в виде стека.

В старших адресах в порядке убывания адресов находятся области: сохранения регистров, списка параметров, программы пункта задания, таблицы ввода-вывода задачи (TIOT). Создание и заполнение этих областей производит программа планировщика. Данные области памяти остаются занятыми в течение всего времени выполнения пункта задания.

Распределение памяти внутри раздела в режимах MFT и MVT в целом совпадает. Однако в режиме MVT для устранения фрагментации имеется подпул, состоящий из соседних блоков памяти размером 2 К с номерами от 0 до 255. Для каждого запроса выделяется подпул с определенным номером.

Программы пункта задания обращаются к супервизору с явными или неявными запросами. Явные запросы осуществляются по макрокоманде GETMAIN, освобождение памяти происходит по макрокоманде FREEMAIN. Неявные запросы осуществляются по макрокомандам, не требующим для выполнения своих функций выделения памяти (LINK).

При выделении областей памяти для пункта задания по макрокоманде GETMAIN ее адрес передается программе, которая сообщает в запросе длину требуемой области. Память отводится с границы двойного слова, освобождается по завершении пункта задания либо выполнении макрокоманды FREEMAIN.

Рассмотрим функции макрокоманды GETMAIN, которая позволяет осуществлять следующие четыре типа запросов: регистровый (R), элементарный (E), списковый (L), переменный (V). Тип запроса отмечается буквой первого оператора. Регистровый формат макрокоманды имеет вид

$$[\text{имя}]\ \text{GETMAIN}\ \ R,\ LV=\left\{\begin{matrix}\text{число}\\ 0\end{matrix}\right\}[,\ SP=\text{число}]$$

Параметр LV определяет длину запрашиваемой области в байтах, кратную восьми. При регистровом запросе длина может быть указана числом в регистре 0.

Параметр SP определяет номер подпула (0–127), из которого отводится участок памяти (по умолчанию 0).

Пример. Пусть необходимо получить область памяти размером 320 байт, адрес которой поместить в регистр 5:

```
GETMAIN  R, LV = 320
LR       5, 1
```

Освобождение памяти выполняется макрокомандой FREEMAIN. Для примера, приведенного выше, освобождение памяти выполнится по МК:

```
FREEMAIN  R, LV = 320
```

Управление обменом страниц. Виртуальная память в ОС ЕС делится на страницы по 2 или 4 К байт. Страницы могут находиться в основной памяти или во внешней – в страничном наборе SYS1.PAGE.

При обращении к виртуальному адресу страницы, находящейся во внешней памяти, происходит прерывание, по которому вызывается супервизор страниц. Последний пересылает требуемую страницу из внешней в основную память. Он ведет учет расположения страниц и при высокой частоте обмена страницами уменьшает число выполняемых заданий приостановкой низкоприоритетных.

Преобразование виртуальных адресов в реальные осуществляет устройство динамического преобразования адресов, которое получает требуемую информацию от супервизора страниц. Для обмена страниц в памяти создаются таблицы сегментов и страниц. В сегмент объединяются 32 или 16 страниц (в зависимости от размера страниц, расположенных последовательно).

В режиме SVS число разделов для выполнения заданий не ограничено. Это связано с тем, что для раздела в памяти хранятся не все таблицы страниц, а только требуемых сегментов.

Виртуальный адрес включает индексы сегмента страницы и байта. Реальный адрес состоит из номера страницы и индекса байтов. Для преобразования виртуального адреса в реальный происходят два обращения к основной памяти. Во время первого из таблицы сегментов считывается таблица страниц, а по ней при втором обращении определяется номер страницы, дающий с индексом байта требуемый адрес. Это преобразование выполняется при наличии страницы в памяти. При ее отсутствии в результате прерывания супервизор страниц перемещает в память нужную страницу и корректирует таблицу страниц. Затем прерванная программа производит преобразование виртуального адреса в реальный. Если свободной страницы в оперативной памяти нет, супервизор страниц производит замену, освобождая некоторую страницу путем перемещения ее во внешнюю память. Не пересылается страница, содержимое которой со времени переноса не изменилось.

2.5. УПРАВЛЕНИЕ ЗАДАЧАМИ

Блок управления задачами. В процессе инициирования для каждого пункта задания создается одна задача. В режимах MFT, MVT, SVS с подзадачами основная задача может образовывать дополнительные. Преимуществом подзадач является возможность передачи процессора от основной задачи раздела к подзадаче, т. е. параллельная обработка. Информация о каждой задаче содержится в блоке управления задачами (TCB). Этот блок создается в момент образования задачи из пункта задания или возникновения подзадачи по МК ATTACH.

Блок TCB представляет собой таблицу, которая содержит адреса управляющих блоков и таблиц задачи, TCB следующих задач в очереди, TCB образующей и образованных задач таблицы ввода-вывода и другую информацию.

Количество блоков TCB в системе определяется числом задач, одновременно выполняемых в режиме мультипрограммирования. В момент образования задачи TCB заносится в очередь задач, а в момент завершения удаляется из нее. Блоки TCB в очереди упорядочены по убыванию диспетчерского прио-

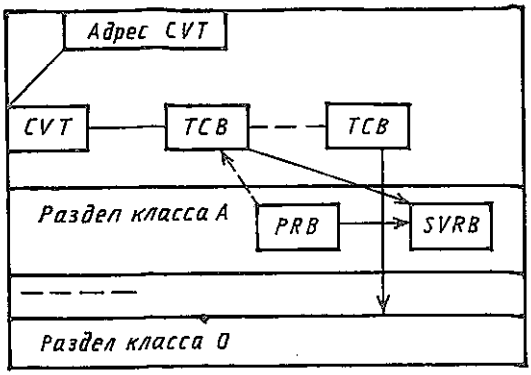


Рис. 2.4.
Очередь блоков управления задачами.

ритета, который указывается параметром DPRTY оператора EXEC. Ресурсы системы распределяются для задач также в порядке их диспетчерских приоритетов.

В системе, кроме проблемных, могут находиться системные задачи (связи, ПЛАНИРОВЩИК, обработки ошибок ввода-вывода и т. д.). Для получения ресурсов эти задачи имеют наивысший приоритет. Центральным процессором управляет задача, находящаяся в состоянии готовности и имеющая наивысший диспетчерский приоритет. Она определяется программой супервизора-диспетчера, который получает управление при обработке прерывания. Диспетчер просматривает список готовых задач и переводит первую из них в состояние активности.

Присутствие в памяти раздела активной программы указывается блоком запроса программы. Если в границах данного раздела обрабатываются прерывания по обращению к супервизору, об этом свидетельствует присутствие в памяти блока запроса супервизора (SVRB). Блоки запроса определяют активные модули, исполняемые в данном разделе. Если очередь блоков запросов пуста, то пуст и соответствующий раздел. Одной из функций инициатора является стирание после завершения задачи соответствующих ей блоков запроса.

Ключом к поиску системных управляющих блоков и указателей является таблица вектора связей (CVT) , которая содержит адреса большей части важнейших блоков управления. Адрес CVT находится в поле третьего двойного слова основной памяти (рис. 2.4) .

В режиме с фиксированным числом разделов без подзадач каждая задача располагается в одном разделе. Поскольку известно количество системных задач, длина очереди TCB фиксирована.

При работе с подзадачами в режиме с фиксированным числом разделов очередь TCB сначала также фиксирована и расположена в ядре. При образовании подзадачи для нее строится блок TCB в области системных очередей, который помещается в очередь в соответствии со своим приоритетом.

В режиме с переменным числом задач количество разделов переменное. Диспетчерский приоритет определяется параметром DPRTY оператора EXEC и может меняться макрокомандой CHAP. Блоки TCB для всех задач, кроме системных, содержатся в области системных очередей.

Основные программы супервизора задач. Очереди TCB в системе соответствует таблица задач, каждая строка которой описывает один блок управления задачей. На основании этой таблицы организуется инициирование задачи, выделение ей ресурсов, формирование очередей задач, их синхронизация и управление выполнением.

Все программы обработки прерывания передают управление диспетчеру, основное назначение которого заключается в выборе из таблицы задач системной или проблемной задачи с наивысшим приоритетом.

Приведем укрупненный алгоритм диспетчера:

1. Если есть прерывания, управление передается одной из программ супервизора прерываний в зависимости от признака в соответствующем регистре.

2. При отсутствии прерываний и наличии системных задач управление передается задаче с высшим приоритетом.

3. При отсутствии системных задач ищется проблемная задача с высшим приоритетом. Для найденной задачи (в том числе системной) загружается ССП и управление передается выбранной программе.

4. Если эта программа последняя, то не сохраняются регистры подчиненной программы, поскольку их содержимое не изменено.

5. При отсутствии проблемных задач супервизор бездействует.

Макрокоманды супервизора. Основные программы супервизора работают с рядом макрокоманд (МК), которые можно разделить на следующие группы: управления связями, подзадачами, ресурсами, синхронизации событий и работы с таймером.

Рассмотрим макрокоманды *управления связями*. В программах простой и оверлейной структур функции супервизора заключаются в поиске и загрузке в память нужных сегментов выполняемых команд. При этом отсутствует необходимость использования SVC-команд для работы с загрузочными модулями. Вызов сегментов может производиться по макрокоманде CALL , имеющей следующий формат:

$$[\text{имя}]\ \text{CALL}\ \left\{\begin{matrix}\text{EP=n}\\(15)\end{matrix}\right\}\ [,(\text{PARAM})\,[,\text{VL}]]\,[,\text{ID}=\text{число}]$$

Имя точки входа указывается параметром EP или засылается в регистр 15. В операнде PARAM указываются имена передаваемых параметров. Если указывается VL , адрес последнего параметра содержит единицу в старшем бите. ID используется для формирования адреса возврата (при его отсутствии возврат управления передается команде, следующей за CALL). Для передачи управления может применяться макрокоманда RETURN, в которой указываются область восстановления и информация о коде возврата.

В динамических структурах передача управления связана с поиском программы в библиотеке и загрузкой ее в память. Для поиска и загрузки элемента оглавления библиотеки может использоваться макрокоманда BLDL. Макрокоманды LINK, LOAD, XCTL служат для загрузки вызываемого сегмента в память.

Макрокоманда LINK осуществляет передачу управления с возвратом и имеет формат

$$[\text{имя}\ ]\ \text{LINK}\ \left\{\begin{matrix}\text{EP=n}\\\text{EPLOC=A1}\\\text{DE=A2}\end{matrix}\right\}\ \begin{matrix}[,\text{DCB=адрес}]\\[,\text{PARAM=(A, ...)}]\,[,\text{VL=1}]\,[,\text{ID=число}]\end{matrix}$$

EPLOC=A1 указывает адрес точки входа, DE=A2 определяет адрес поля имени точки входа, используется после МК BLDL. Параметр DCB задает адрес блока управления данными библиотеки, содержащей требуемую программу. Смысл параметров EP, VL, ID аналогичный, как и в МК CALL.

По макрокоманде LOAD в память загружается модуль и остается там до удаления макрокомандой DELETE или завершения задания. Формат макрокоманды LOAD

$$[\text{метка}]\ \text{LOAD}\left\{\begin{matrix}\text{EP=n}\\ \text{EPLOC=A1}\\ \text{DE=A2}\end{matrix}\right\}\ [,\text{PARAM=(A)}]\ [,\text{VL=1}]\ [,\text{ID=n}]\ [,\text{DCB=адрес}]$$

Смысл параметров МК LOAD такой же, как и в МК LINK.

Макрокоманда DELETE уменьшает на единицу счетчик обращений для программы с указанной точкой входа. При нуле счетчика программа удаляется из памяти. Формат макрокоманды

$$[\text{метка}]\ \text{DELETE}\ \left\{\begin{matrix}\text{EP=n}\\ \text{EPLOC=A1}\\ \text{DE=A2}\end{matrix}\right\}$$

Рассмотрим пример загрузки программы A (макрокоманда M0), обращения к ней (макрокоманды M1, M2) и удаления ее из памяти (макрокоманда M3):

```
M0      LOAD        EP=A
...
M1      LINK        EP=A, PARAM=(X, Y, Z), VL=1
...
M2      LINK        EP=A, PARAM=(W, Y, Z), VL=1
...
M3      DELETE      EP=A
...
W       DC...
X       DC...
Y       DC...
Z       DC...
```

К другим макрокомандам для управления связями относятся SAVE, FETCH, IDENTIFY. Макрокоманда SAVE используется для сохранения содержимого регистров вызывающей программы в области сохранения, адрес которой запоминается в 13-м регистре.

Макрокоманда выбора FETCH помещает в память модуль, выполняя при этом подготовительные операции: считывает записи модуля и устанавливает адресные константы, зависящие от расположения. Макрокоманда IDENTIFY указывает имя дополнительной точки входа в модуль и ее адрес.

Для *управления подзадачами* используются макрокоманды ATTACH и DETACH. По первой МК создается подзадача, которой передается управление. Подзадача может возвращать управление на время ее прерывания, т. е. организуется псевдопараллельное выполнение. В регистр 1 передается адрес TCB образованной подзадачи. Формат макрокоманды

```
                  ⎧EP=n     ⎫
[метка] ATTACH    ⎨EPLOC=A1 ⎬  [,DCB=адрес]     [, LPMOD=n1]
                  ⎩DE=A2    ⎭  [,DPMOD=n]       [,PARAM=(A,,)] [,VL=1]
                                [,ECB=адрес ECB]  [,ETXR=адрес выхода]
```

Параметры EP, EPLOC, DE, DCB, PARAM, VL имеют те же значения, что и для МК LINK.

Параметры LPMOD и DPMOD указывают число без знака, которое нужно вычесть или прибавить из граничного или текущего приоритета образующей задачи для получения граничного (текущего) приоритета подзадачи. В операнде ECB задается адрес блока управления событием. Операнд ETXR определяет адрес вспомогательной программы для получения управления по завершению вызываемой задачи. Макрокоманда DETACH удаляет указанную подзадачу из системы.

Пример. В программе выдается МК ATTACH, которая образует подзадачу B. Адрес блока управления подзадачи запоминается в блоке TB, который позже используется в МК DETACH для уничтожения подзадачи:

```
     .............
     ATTACH EP=B,PARAM=(ECB1,P1),VL=1,DPMOD=1,ECB=E4
     ST     1,TB
     .............
DTB  DETACH TB
     .............
 TB  DS   F
 E4  DS   F'0'
FCB1 DS   F'0'
 P1  DS...
```

Если в МК ATTACH не заданы операторы ECB, ETXR (указание на нормальное или аварийное завершение подзадачи), то управляющая программа такую подзадачу исключает автоматически после завершения без МК DETACH.

Для изменения текущего приоритета задачи либо подзадачи используется МК CHAP, которая имеет формат

```
                                  ⎡⎧адрес   адреса TCB⎫⎤
[метка]  CHAP  n                  ⎢⎨                  ⎬⎥
                                  ⎣⎩      'S'         ⎭⎦
```

Величина n — изменение приоритета — складывается с текущим приоритетом; адрес адреса TCB — адрес слова, содержащего адрес TCB задачи. Новое значение приоритета может превышать граничное.

Для *управления ресурсами* (программы, область памяти, файлы), повторные запросы на которые задерживаются до завершения работы с ними, используются МК ENQ и DEQ. По МК ENQ формируется запрос на ресурс, а по DEQ производится его освобождение. В качестве параметров задаются имя очереди к ресурсу, имя ресурса, область и режим использования, вид запроса (условный и безусловный) и т. д.

Пример. Несколько программ могут повторно использовать программу P. Для предотвращения одновременного использования программы P выдается макрокоманда ENQ, а после обращения – макрокоманда DEQ:

```
ENQ (N, M)
CALL P
DEQ (N, M)
. . . . . .
NDC  CL8' Q1'
MDC  CL1' P'
```

Для синхронизации событий при разбиении задачи на подзадачи используются макрокоманды WAIT и POST . Для связи задачи с подзадачами применяется блок управления событием ECB, который создается при выполнении макрокоманд ATTACH. По завершении события (подзадачи) в блок ECB заносится информация о его окончании.

Для перевода задачи в состояние ожидания служит макрокоманда WAIT, которая может указать несколько событий (для каждого из них – блок ECB). Макрокоманда POST устанавливает признак в блоке ECB при завершении события, в ней указываются адрес ECB и код завершения.

2.6. УПРАВЛЕНИЕ ПРЕРЫВАНИЯМИ

Механизм прерываний. Для управления прерыванием используется супервизор прерываний. Он обеспечивает передачу управления от прерванной программы программам обработки прерываний и возвращение к прерванной.

Супервизор прерываний выполняет следующие функции: сохраняет состояние системы в момент прерывания; различает типы прерываний; передает управление программ обработки прерываний, возвращает управление в прерванную программу.

Для реализации механизма прерываний рассмотрим в основной памяти несколько типичных областей: прикладной программы, программы обработки прерывания, текущего ССП, двух двойных слов, содержащих старое и новое ССП.

При выполнении прикладной программы возникает ситуация, когда требуется изменить ход вычислений (ввод-вывод, внешние события и т. д.). В этом случае текущее ССП копируется в область старого, а содержимое нового ССП посылается в область текущего ССП. Процессор по адресу в трех последних байтах текущего ССП отыскивает очередную программу (обработки прерывания) и производит ее выполнение. После этого управление вновь передается прикладной программе пересылкой содержимого старого ССП в поле текущего.

В системе ЕС ЭВМ выделены пять типов прерываний: внешние, при обращении к супервизору (SVC), программные, от схем контроля и ввода-вывода. Для каждого из них в начальной области памяти имеются старое и новое ССП. Последнее передает управление соответствующему обработчику прерываний, входящему в состав супервизора прерываний.

Для последовательной обработки прерываний используется *маскирование* в отдельных битах ССП (ввода-вывода и внешние разряды 0–7, от схем контроля – 13, программные – 36–39). Одновременно SVC и программные прерывания не возникают, поскольку в каждый момент времени процессор выполняет одну команду. При одновременном возникновении остальных прерываний сначала обрабатываются прерывания от схем контроля, поскольку не име-

ет смысла работать супервизору в неисправной машине.

При отсутствии схемных прерываний происходит следующее: в старых ССП запоминаются соответственно содержимое ССП прикладной программы (по программному прерыванию), содержимое ССП обработки программных прерываний (по внешнему прерыванию), содержимое ССП внешнего прерывания (по прерыванию ввода-вывода). Выполняется программа супервизора обработки прерываний ввода-вывода. По ее завершении в поле текущего ССП пересылается содержимое старого ССП ввода-вывода и управление передается обработчику внешних прерываний. После этого снова в поле текущего ССП пересылается старое ССП внешних прерываний, по которому управление передается обработчику программных прерываний. Вместо программных прерываний могло быть прерывание по SVC, но последовательность обработки прерываний сохраняется. Порядок убывания их приоритетов таков: от схем контроля, по вводу-выводу, внешние, программные или по SVC.

Обработка программных прерываний. Программные прерывания возникают из-за неверного использования команд и данных. В ЕС ЭВМ различают 15 типов программных прерываний, каждому из которых соответствует свой код прерываний (табл. 2.2).

Программные прерывания с кодами 8, 10, 13, 14 можно замаскировать при помощи команд SPH (установить маску системы).

Обработчик программных прерываний планирует аварийное завершение задачи, при которой возникло программное прерывание. Если это подзадача, то аварийно завершается весь пункт задания. Однако, когда это нежелательно, программист может составить свою программу обработки прерываний и сообщить об этом супервизору макрокомандой SPIE. Программное прерывание, возникшее в такой программе, всегда завершается аварийно. Действие макрокоманды SPIE распространяется до такой макрокоманды, которая может либо отменить действие предыдущей, либо указать новую программу обработки прерываний. Формат макрокоманды

[метка] SPIE [A, (n)]

A, указывая адрес программы обработки прерываний, используется для передачи управления, если возникшие прерывания имеются в списке n в виде десятичного числа (табл. 2.2).

Управление таймером. Наиболее частым случаем внешних прерываний является прерывание от таймера (системных часов). Для работы с таймером используются макрокоманды TTIMER и STIMER.

Макрокомандой STIMER задаются интервалы времени, по окончании которых можно запланировать выход в специальную программу. Отсчет времени может производиться непрерывно либо только в моменты активности программы, выдавшей макрокоманду. Возможно переведение задачи в режим ожидания до истечения заданного интервала.

Макрокоманда TTIMER может выдать время, оставшееся до конца интервала, либо отменить действие установленного ранее интервала времени. Ее формат

[метка] TTIMER [cancel]

Любая задача в каждый момент времени может иметь один интервал; очередная макрокоманда STIMER отменяет действие предыдущей. Но в тай-

Таблица 2.2

Причина прерываний	Код	Причина прерываний	Код
Некорректность кода операции	1	Переполнение в десятичных операциях	10
Некорректное использование привилегированной команды	2	Некорректность деления в десятичных операциях	11
Некорректное использование команды ВЫПОЛНИТЬ (EX)	3	Переполнение порядка в операциях с плавающей запятой	12
Нарушение защиты памяти	4	Исчезновение порядка в операциях с плавающей запятой	13
Неправильная адресация	5	Потеря значимости в операциях с плавающей запятой	14
Неправильная спецификация	6	Некорректность деления в операциях с плавающей запятой	15
Неправильные данные	7		
Переполнение в операциях с фиксированной запятой	8		
Некорректность деления в операциях с фиксированной запятой	9		

мер каждый раз помещается не величина интервала, а разность двух соседних интервалов.

Параллельно с обслуживанием запросов на отсчет интервалов супервизор программно следит за содержимым байта памяти, называемого псевдочасами. Величина каждого нового интервала таймера добавляется к содержимому псевдочасов, поэтому в них все время присутствует момент окончания очередного интервала.

Выводы

1. Вычислительные процессы на внешнем уровне в ОС ЕС организуются при разработке программ, управлении прохождений заданий через систему и управлении работой ОС. На логическом уровне организация вычислительных процессов связана с управлением ресурсами (процессором, памятью, внешними устройствами, программами и данными), контролем функционирования системы и ее восстановлении при сбоях.

2. Основным средством управления вычислениями для пользователя в ОС ЕС является язык управления заданиями. Оператор может управлять функционированием системы, используя директивы командного языка.

3. На логическом уровне управление в ЕС ЭВМ осуществляется супервизором, который на уровне ядра включает ряд функций. Последние обеспечивают работу с памятью, в том числе и с виртуальной, связь модулей, работу с задачами (процессами), распределение ресурсов, связь с оператором, работу с таймером. Вызов функций супервизора поддерживается механизмом прерываний. Для работы программиста с функциями супервизора имеется набор макрокоманд ассемблера.

4. Управление памятью в ОС ЕС осуществляется в трех режимах: с фиксированным числом разделов (MFT), с переменным числом разделов (MVT) и с виртуальной памятью (SVS). Динамическое распределение и освобождение памяти реализуется макрокомандами.

5. Для управления задачами используется супервизор задач, который хранит информацию о состоянии всех задач, определяет, какие программы необходимо запустить. Управляющая информация для каждой задачи находится в таблице TCB. В таблице векторов связей хранится адрес первой TCB, в которой содержится адрес второй, и т. д.

6. Механизм прерываний в ОС ЕС для каждого из пяти видов прерываний обеспечи- ии-
вает сохранение старого ССП и выборку нового из фиксированных ячеек памяти. ССП содержит адрес возврата, код прерывания и другие характеристики прерванной программы.

Рекомендуемая литература: [2, 6, 8, 10].

Вопросы для контроля

1. Назовите состав ПО ЕС ЭВМ.
2. Укажите назначение управляющих и обрабатывающих программ ОС ЕС ЭВМ.
3. Объясните различие между последовательной и параллельной динамическими структурами программ.
4. Поясните назначение основных операторов ЯУЗ: а) JOB; б) EXEC; в) DD.
5. Какие компоненты ОС участвуют при трансляции задания?
6. Какие компоненты входят в состав супервизора ОС ЕС?
7. Как супервизор получает управление?
8. Какие макрокоманды осуществляют управление связями, подзадачами?
9. Какие макрокоманды управляют памятью?

3. ОРГАНИЗАЦИЯ И УПРАВЛЕНИЕ ДАННЫМИ

3.1. ОРГАНИЗАЦИЯ ДАННЫХ В ОПЕРАЦИОННЫХ СИСТЕМАХ

Записи. Основной единицей данных для ОС является набор данных, имеющий имя и строго определенную организацию и состоящий из логических записей. Логическая запись – это единица данных, с которой работает программист. На внешнем носителе наборы данных размещаются в виде блоков.

Набор данных представляет совокупность данных, имеющих логическую и физическую организацию. Организация, не зависящая от природы физического носителя и определяемая программистом, называется *логической*, а организация данных на носителях – *физической*.

В логической организации выделяют следующие единицы: поле данных (домен), логическую запись (строку), набор данных (совокупность строк). Логические записи могут иметь один из трех форматов: фиксированной длины (F), переменной (V) и неопределенной (U).

Для записи переменной длины вначале указывается ее размер. Для записи неопределенной длины конец определяется по специальному признаку – разделителю записи.

Данные на магнитных лентах (МЛ) или магнитных дисках (МД) разделяются на физические записи. Характерным для физической записи является ее чтение или запись за одно обращение к внешнему устройству. Физической записью будет строка печатающего устройства, зона магнитной ленты, сектор гибкого магнитного диска.

Логическая и физическая записи могут не совпадать по размерам. Объединение нескольких логических записей в одну физическую (блок) называется *блокированием*, а записи при этом считаются *сблокированными*. Процесс выделения логических записей из блока носит название *деблокирования*.

Блокирование записей. Количество логических записей в каждой физической называют *коэффициентом блокирования* ($K_б$). При $K_б = 1$ записи не сблокированы. Записи фиксированной длины группируются в блоки (рис.3.1, *а*). При $K_б = \text{const}$ блок содержит одинаковое количество записей (рис.3.1,*б*).

Логические записи переменной длины (V) имеют указатели длины (RL), а при объединении в блоки – указатели длины блока (BL) (рис. 3.2, *б*). В несблокированных записях переменной длины логическая запись и управляющая информация блока составляют физический блок (рис. 3.2, *а*).

Для носителей информации размер блоков ограничен. На МД он не может превышать размера дорожки, если отсутствуют средства переполнения дорожки. На МЛ можно записать блоки максимальной длины до 32 760 байт.

Достоинства блокирования заключаются в уменьшении операций ввода-вывода за счет считывания целиком одного блока (несколько логических записей) и уменьшении объема внешней памяти, занятой набором данных, за счет сокращения межзонных промежутков (25–30 мм), которые необходимы для запуска и останова ленты.

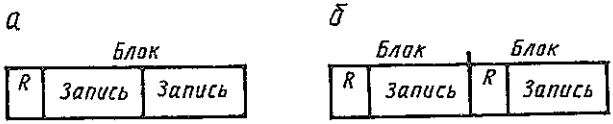


Рис. 3.1.
Записи фиксированной длины.

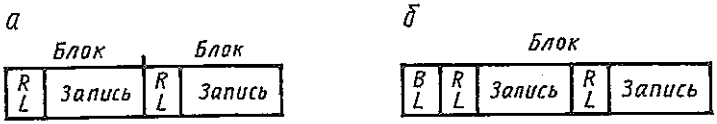


Рис. 3.2.
Записи переменной длины.

При выполнении операций ввода-вывода обмен между внешней и основной памятью осуществляется блоками. Для размещения блока в основной памяти отводится область, называемая *буфером*. Буфера бывают входными и выходными.

К недостаткам буферизации относится необходимость включения программных средств для блокирования и разблокирования записей.

Одному набору данных может соответствовать несколько связанных буферов, которые называются *буферным пулом*. Пул включает блок управления, содержащий адрес первого буфера, число буферов и длину буфера. Затем следуют связанные буфера.

Организация наборов данных. В современных ОС различают четыре основных типа организации наборов данных (НД): последовательная, индексно-последовательная, библиотечная и прямая.

Последовательная организация характеризуется тем, что записи располагаются друг за другом. При обработке последовательного набора данных для достижения некоторой записи необходимо просмотреть все предшествующие ей. Обычно в последовательных наборах записи упорядочиваются по некоторому признаку – ключу в процессе сортировки, а затем над полями записей могут выполняться операции.

Последовательные наборы данных могут состоять из записей фиксированной и переменной длины. Они могут располагаться на различных физических устройствах: ввод-вывод с перфокарт, АЦПУ, МЛ, МД, причем задать конкретное устройство для обработки последовательного набора можно в момент решения задачи средствами ЯУЗ в операторах DD.

При *индексно-последовательной организации* наборы данных могут состоять из записей фиксированной и переменной длины. Записи в наборе упорядо- [illegible] Ключ блока является максимальным из ключей всех записей. Следо[illegible] требуемой записи нет необходимости сравнивать его со[illegible] сей, а лишь с ключом блоков. Индексно-последовател[illegible] ются на томах прямого доступа. При этом создаются т[illegible] чей): дорожки, цилиндра и главный индекс, для чего [illegible] циальные области. Блоки записей размещаются на дор[illegible] для каждого цилиндра генерируется индекс дорожек.

жек имеется элемент, который наряду с адресом дорожки содержит максимальный ключ дорожки (рис. 3.3). Элемент индекса цилиндра содержит максимальный ключ записи, расположенной на данном цилиндре.

Каждый набор данных делится на три области: основных данных, области переполнения и индексов. *Область основных данных* содержит рассортированные записи и может быть заполнена частично или полностью. При частичном заполнении новые записи помещаются в основную область на каждом цилиндре. *Область переполнения* (ОПр) цилиндра содержит записи, которые оказались вытесненными со своих дорожек или не поместились при новой записи. В этой области записи располагаются в несблокированном формате и связаны с помощью указателей с основными данными. В общей области переполнения могут размещаться записи, для которых не нашлось места в области переполнения цилиндра.

При поиске данных просматривается сначала главный индекс, затем индекс цилиндров и, наконец, индекс дорожек (ИД). В режиме обновления возможны операции: последовательного поиска записи в блоке, произвольного поиска записи в любом участке файла, добавление записей в файл.

Последовательный поиск означает чтение записей из области основных данных. Если существуют записи переполнения, то они выбираются при помощи указателей цепочки. Для этого в индексе дорожки имеется указатель переполнения. Произвольный поиск осуществляется посредством использования всех индексов, т. е. сначала определяется индекс цилиндров, затем дорожек, и, наконец, поиск производится внутри дорожки.

Механизм добавления новых записей заключается в следующем (рис. 3.4). Новая запись с ключом *5* помещается в существующую, и происходит сдвиг записей, и запись с ключом *25* оказывается в области переполнения. При этом записывается указатель переполнения, однако индекс блока не меняется. При добавлении записи с ключом *27* перемещения в основной области не происходит. При этом модифицируются указатели связанного списка в области переполнения и старший индекс дорожки.

Библиотечная организация используется для работы с наборами данных на томах прямого доступа. Она состоит из последовательно организованных наборов (разделов) и их справочника (каталога), в котором записываются имена и адреса разделов (рис. 3.5).

Память для оглавления библиотеки выделяется в момент создания файла. Размер памяти для оглавления указывается в параметре SPACE оператора DD, в котором задается число блоков оглавления. В системе ОС ЕС имеется ряд программ для создания и реорганизации библиотек, распечатки оглавлений и разделов, переименования и удаления разделов и т. д.

Прямая организация может использоваться на томах прямого доступа и позволяет найти требуемую запись среди произвольно расположенных. При этом записи, предшествующие или следующие за искомой, не считываются. Это достигается за счет того, что программист разрабатывает механизм вычисления физического адреса записи по ключу, который используется при чтении записи.

В этом случае имеется набор данных с записями фиксированной длины, блок содержит одинаковое число записей, а дорожка – блоков. При этом по номеру записи легко вычислить ее место на дорожке.

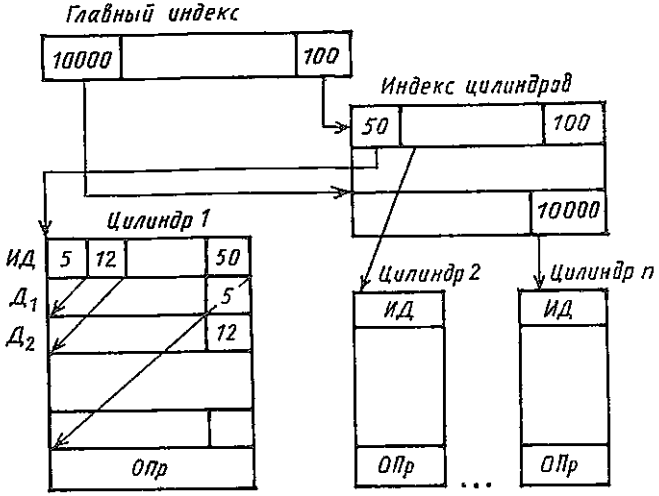


Рис. 3.3.
Структура индексно-последовательного набора данных.

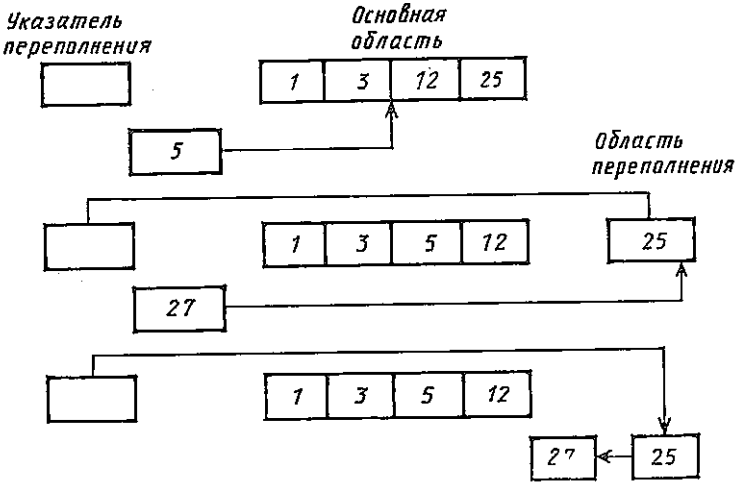


Рис. 3.4.
Добавление новых записей в индексно-последовательный НД.

При косвенной адресации адрес каждой записи получается в результате выполнения некоторой операции над ключом, которая называется хэшированием (hashing). Алгоритм хэширования выполняется в два этапа: 1) если ключ не цифровой, он преобразуется в соответствующее цифровое представление; 2) цифровой ключ преобразуется в произвольное число (адрес).

На практике существует много алгоритмов хэширования, которые должны обеспечивать преобразование ключа в адрес заданного диапазона и минимальное совпадение адресов [8].

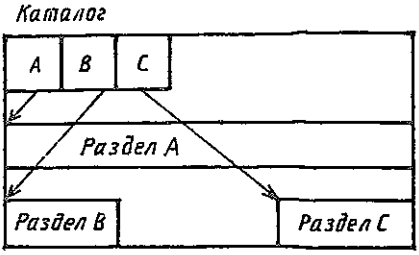


Рис. 3.5.
Структура библиотечного набора данных.

Программы управления данными реализуют различные способы организации данных и выполняют операции обмена между основной памятью и внешними устройствами. Обмен данными осуществляется программами методов доступа, принятых в ОС ЕС, и супервизором ввода-вывода.

Организация данных на магнитной ленте. В простейшем случае НД представляет собой совокупность блоков, расположенных между двумя ленточными маркерами. Такой режим задается управляющей программе в параметре LABEL оператора DD как режим работы без меток, однако он может привести к ошибкам оператора или программиста. Другой организацией данных на МЛ является использование стандартных меток, которые представляют собой физические записи длиной 80 байт. В этом случае информация на МЛ имеет следующую структуру: группу меток тома, наборы данных с метками начала набора, собственно данные и группу меток конца НД (или конца тома). На МЛ возможна лишь последовательная организация НД.

Организация данных на томах прямого доступа. Рассмотрим организацию данных на пакетах дисков. Информация располагается на дорожках. Однопозиционные дорожки для механизма доступа накопителя образуют цилиндр. Цилиндры и дорожки внутри его нумеруются, их число зависит от пакета дисков. Запись информации на дорожку производится в виде блоков. Физический адрес блока определяется как ССННР, где СС – номер цилиндра, НН – номер дорожки, Р – номер блока. Блоки записываются на дорожку в виде областей: идентификатора, ключа и данных, разделенных промежутками. Область ключа может отсутствовать, поэтому блоки могут быть с ключами и без них.

Для указания начальной точки записи на всех дорожках пакета имеется метка начала оборота. В начале дорожки записывается ее собственный адрес, затем блок Р, в котором записывается информация о заполнении дорожки данными. Затем следует адресный маркер, идентифицирующий начало каждого блока. Блок начинается с области идентификатора, который содержит адрес блока, а также информацию о длине ключа и данных. Ключ содержит признак блока (имя блока) для его поиска.

3.2. МЕТОДЫ ДОСТУПА В ОС ЕС

Понятие доступа. Способом доступа является вид передачи данных между основной памятью и внешней. В ОС ЕС используются два основных способа доступа: базисный и с очередями. В *базисном способе* передача данных осуществляется блоками, программист выполняет деление блоков на записи, орга-

низует буферизацию данных. В *способе доступа с очередями* буферизация и деление блоков на записи происходят автоматически.

Сочетание способа доступа с организацией набора данных называется методом доступа. Базисный способ доступа применим ко всем организациям наборов данных, способ доступа с очередями – только для последовательной, индексно-последовательной организации. Основные методы доступа ОС ЕС следующие: последовательный метод доступа с очередями QSAM, базисный последовательный метод доступа BSAM, базисный библиотечный метод доступа PBAM, индексно-последовательный метод доступа с очередями QISAM, базисный индексно-последовательный метод доступа BISAM, базисный прямой метод доступа BDAM. Каждый метод доступа имеет свой набор системных макрокоманд.

Кроме основных методов доступа, программист имеет возможность использовать физический метод доступа, а при работе в режиме SVS (виртуальная память) – виртуальный метод доступа. При работе с программами машинной графики применяется графический метод доступа.

Физический метод доступа использует макрокоманду EXCP (выполнение канальной программы), при этом программист должен составить канальную программу и сформировать блоки ввода-вывода (IOB) и управления событиями (ECB). Блок IOB содержит следующие адреса: канальной программы, блока управления данными (DCB), блока управления событиями (ECB), а также области сохранения для слова состояния канала и байта уточненного состояния.

Управляющие блоки. Прежде чем начать обработку набора данных, необходимо сформировать информацию о наборе, характеристиках внешнего устройства и требования к обработке. Эта информация передается управляющей программе через блок DCB. Информация в этот блок заносится из трех источников: из программы пользователя, оператора DD и метки набора данных (только для существующих наборов с меткой). Блок DCB заполняется частично из макрокоманды DCB, выдаваемой программой. Затем он дополняется в процессе выполнения программы при открытии набора данных по макрокоманде OPEN. Это позволяет создавать программы, независимые от устройства ввода-вывода.

Кроме блока DCB, в процессе ввода-вывода используется еще несколько управляющих блоков и таблиц. Блок IOB служит для связи между программой пользователя, которая инициирует ввод-вывод, и супервизором ввода-вывода, непосредственно выполняющим обмен.

Блок управления устройством UCB содержит характеристики отдельного внешнего устройства, которые используются супервизором ввода-вывода и планировщиком заданий для распределения устройств. Блок UCB создается при генерации ОС для каждого внешнего устройства.

Блок ECB применяется супервизором ввода-вывода для сообщения программе пользователя о завершении ввода-вывода. Супервизор помещает в этот блок код завершения, который информирует о завершении выполнения канальной программы (с ошибкой или без нее). Блок ECB строится в программе по макрокоманде запроса на ввод-вывод (GET, READ и др.).

Блок экстентов данных DEB дополняет блок DCB и содержит информацию о томе, наборе данных, его границах на внешнем носителе и методе досту-

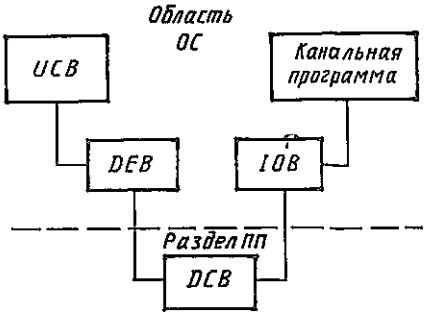


Рис. 3.6.
Связь блоков управления вводом-выводом.

па. Блок DEB включает ссылку на блок управления устройства UCB. Блок DEB создается во время выполнения программы при открытии набора данных. Связь блоков показана на рис. 3.6. Упрощенная схема организации ввода-вывода для программы на языке ассемблера приведена на рис. 3.7. При использовании одного из вышеуказанных методов доступа ОС ЕС программист должен описать блок управления данными DCB одноименной макрокомандой, затем открыть набор данных макрокомандой OPEN и выдать запрос на ввод-вывод макрокомандами GET, PUT (методы доступа с очередями) или READ, WRITE (базисные методы доступа). По одной из последних команд программа метода доступа выполняет следующее: формирует канальную программу на соответствующем устройстве, строит блок ECB, в который по окончании канальной программы заносится код завершения, выдает макрокоманду EXCP для вызова супервизора ввода-вывода и передает ему адрес блока IOB.

Супервизор ввода-вывода инициирует операцию ввода-вывода, а для ее управления выполняет следующие действия: планирует запрос на ввод-вывод; выдает команду начать ввод-вывод (SIO) для запуска канальной программы. В случае ее успешного начала канал и процессор начинают работать параллельно до окончания ввода-вывода. Канал обрабатывает прерывания ввода-вывода и планирует программы коррекции ошибок, помещает код завершения операции ввода-вывода в блок управления событиями ECB.

При обмене данными с использованием базисных методов доступа программист сам должен осуществлять синхронизацию завершения операции ввода-вывода и основной программы. Для этого в соответствующих местах используются макрокоманды WAIT или CHECK. По этим макрокомандам выполнение программы приостанавливается до завершения ввода-вывода. Только после окончания операции ввода-вывода по признаку завершения в блоке ECB может продолжить выполнение программа пользователя.

При использовании физического метода доступа (для создания нестандартных методов доступа) программист сам составляет канальную программу, строит блоки ввода-вывода IOB и управления событиями ECB. Для вызова супервизора ввода-вывода применяется МК EXCP, а для синхронизации обмена – макрокоманда WAIT.

Блок управления данными. Для построения блока DCB служит одноимен-

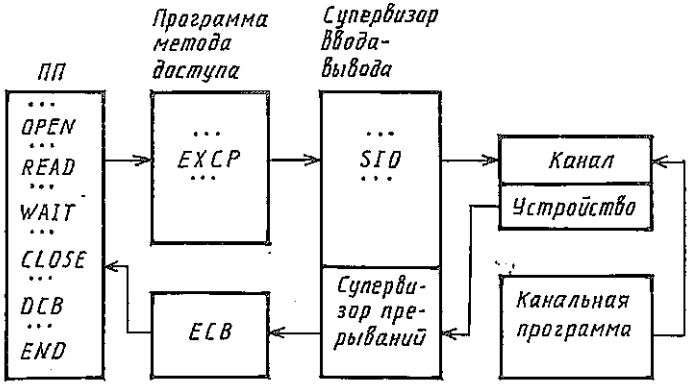


Рис. 3.7.
Схема организации ввода-вывода.

ная макрокоманда, указывающая характеристики наборов данных. Рассмотрим основные операнды, описываемые этой макрокомандой. Обязательными являются операнды: DSORG и MACRF. Тип организации набора указывается с помощью операнда DSORG: последовательная PS, индексно-последовательная IS, библиотечная PO и прямая DA. Если набор данных, записываемый по абсолютному адресу, неперемещаемый, к двум буквам добавляется U, например PSU.

Операнд MACRF указывает макрокоманды, которые будут применяться при работе с наборами данных: MACRF = GM предписывает использование макрокоманды GET в режиме пересылки, а GL – в режиме обработки на месте. Эти два операнда в совокупности определяют метод доступа. Остальные операторы DCB необязательны.

Формат записей задается операндом RECFM: фиксированной длины (F), переменной (V), неопределенной (U). Блокированные записи идентифицируются как FB или VB.

Операнд LRECL задает длину записи в наборе данных. При переменной длине указывается максимальный размер записи.

Операнд BLKSIZE задает максимальную длину блока в байтах. При фиксированном формате длина блока должна быть кратной длине записи, для записей неопределенной длины – длине наибольшего блока, для несблокированных записей переменной длины – значению LRECL + 4.

Операнд BUFNO задает число буферов в буферном пуле (не более 255). Для задания длины буфера используется операнд BUFL (она не должна превышать 32 760).

Операнд DEVD определяет тип устройства для данного НД. Для устройства прямого доступа значение DEVD = DA, для магнитной ленты TA, для АЦПУ PR и т. д.

Для указания действия при возникновении ошибок ввода-вывода используется операнд EROPT. Операнд EODAD задает адрес подпрограммы, на которую передается управление при достижении конца НД.

Операнд SYNAD назначает адрес подпрограммы, на которую передается

управление в случае возникновения постоянной (некорректируемой) ошибки ввода-вывода.

Операнд DDNAME содержит имя оператора DD, идентифицирующего НД, для которого создается блок DCB. Операнд устанавливает связь между НД и блоком DCB при открытии НД, которая разрушается при закрытии.

Виртуальный метод доступа (VSAM). Метод включает два способа организации наборов данных: записи располагаются в порядке возрастания ключей и в порядке их поступления. Записи обоих способов запоминаются в непрерывных областях внешней памяти фиксированной длины – интервалах, которые являются единицей обмена с основной памятью. Размер интервала должен быть кратным размеру физической записи, длина которой в методе VSAM может быть 512, 1024, 2048 и 4096 байт. Пользователь VSAM фактически работает с интервалами.

Адрес записи равен ее смещению от начала набора данных (RBA). В методе используются записи фиксированной и переменной длины. Доступ к записям, упорядоченным по ключу, возможен по значению ключа или смещению RBA, доступ к неупорядоченным записям – только по смещению RBA. Организация НД в порядке возрастания ключей аналогична индексно-последовательной организации, а в порядке поступления записей – последовательной и прямой организации.

При выполнении запроса обрабатывающей программы на запись в виртуальную память считывается интервал данных, включающий эту запись, если его нет в ОП. Метод VSAM указывает длину записи и помещает ее в рабочую область программы или предоставляет программе адрес записи в буфере ввода-вывода VSAM. Возможен доступ не только к записи, но и к интервалу данных, однако при этом усложняется работа программиста.

3.3. МАКРОКОМАНДЫ УПРАВЛЕНИЯ ДАННЫМИ

Чтение, запись наборов данных. При способе доступа с очередями передача данных выполняется макрокомандами GET и PUT. Макрокоманда GET осуществляет выборку логических записей из НД и помещает в заданную область памяти. В зону ввода доставляется очередной блок записей и определяется адрес следующей записи. Автоматически происходит контроль за концом блока и набора данных. Используются три режима обработки. В *режиме указания* запись не перемещается. По МК GET в регистр 1 выдается адрес новой записи во входном буфере, по МК PUT в регистр 1 – адрес для вывода из буфера, куда можно поместить очередную запись. В *режиме пересылки* логическая запись пересылается либо из входного буфера в рабочую область (GET), либо из рабочей области – в буфер вывода (PUT). В *режиме подстановки* логическая запись не пересылается, адрес входного буфера подменяется адресом рабочей области или адрес рабочей области – адресом буфера вывода. Формат МК GET

[метка] GET адрес – DCB [, адрес области]

где адрес DCB – адрес блока DCB для НД, из которого выбирается запись. Адрес области указывается для режима пересылки и является адресом поля, в которое пересылается следующая запись.

Формат МК PUT аналогичен GET с повторением смысла параметров. В базисных методах доступа обмен осуществляется посредством МК READ и WRITE, которые обрабатывают не записи, а блоки. Макрокоманда READ выбирает блок из набора данных и помещает его в указанную область памяти. Успешное окончание ввода проверяется МК CHECK, которая анализирует блок управления событием данных.

Открытие и закрытие наборов данных . После трансляции и редактирования программы необходимо подключить к ней программы, связанные с управлением данных. Это осуществляется МК OPEN. После завершения операции ввода-вывода блок DCB следует восстановить, освободить основную память и создать метки НД, что осуществляется МК CLOSE. Эти две макрокоманды можно одновременно использовать для нескольких НД. Формат макрокоманды открытия НД OPEN

[метка] OPEN (адрес DCB [, (режим [, диспозиция]) ...])

где адрес DCB – адрес МК DCB; режим – режим обработки НД; диспозиция – состояние тома в момент переключения. При отсутствии режима принимается ввод INPUT , при отсутствии диспозиции – ее значение из оператора DD.

Основные режимы обработки означают: INPUT – входной НД; OUTPUT – выходной НД, INOUT – сначала входной, а затем без повторного открытия выходной НД; OUTIN – выходной, а затем без повторного открытия входной НД.

Операнд "диспозиция" указывает, в какую позицию будет установлен том, если при обработке НД обнаружится конец тома: LEAVE – том остается в прежнем положении; REREAD – том устанавливается в такую позицию, чтобы НД можно было снова обработать; DISP – выбирается диспозиция тома, указанная параметром DISP в операторе DD .

По макрокоманде закрытия НД CLOSE образуются выходные метки, производится заданная диспозиция тома. Блок DCB восстанавливается, его значения становятся такими, как до открытия НД, происходит логическое отсоединение набора от обрабатывающей программы. Формат команды закрытия

[метка] CLOSE (адрес DCB , [диспозиция], ...) [TYPE=T]

Если задан параметр TYPE=T, выполняется указанная диспозиция тома (кроме DISP), и блок DCB модифицируется для обработки набора данных без повторного открытия макрокомандой OPEN. Значение остальных параметров аналогично параметрам макрокоманды OPEN.

Рассмотрим пример построения последовательного НД из 500 записей фиксированной длины по 120 байт. Описание НД содержится в макрокоманде DCB. Происходит открытие НД в режиме вывода, занесение его размера в регистр 5:

```
        OPEN    (OUTDCB, (OUTPUT))
        LA      5,500
C1      WRITE   BLOK, SF, OUTDCB, BUF
        CHECK   BLOK
```

```
          BCT     5, C1
          CLOSE   (OUTDCB)
          . . . . .
BUF       DC      200'ДАННЫЕ'
OUTDCB    DCB     DSORG=PS,MACRF=W,DDNAME=E1,BLKSIZE=120,RECFM=F
```

Формирование набора данных осуществляется по МК WRITE, в которой указано имя блока DECB BLOK; режим обработки в прямом направлении SF; адрес блока DCB OUTDCB и адрес буфера, в котором находятся данные, BUF. Макрокоманда CHECK используется для контроля за завершением операции вывода. После записи всего НД происходит его закрытие.

Управление буферами. В ОС ЕС предусмотрено несколько способов построения буферных пулов (набор связанных буферов) для нужд того или иного НД. После построения буферного пула все запросы на буфера, необходимые для НД, удовлетворяются из этого пула.

Различают входные и выходные буфера. Входному НД приписываются входные буфера, выходному НД – выходные. Данные с внешних носителей попадают во входной буфер, из него передаются программе. Данные из программы попадают в выходной буфер, из него – на внешний носитель.

Буферный пул можно построить четырьмя способами. Первый заключается в построении пула в прикладной программе и заполнении соответствующего поля в блоке DCB. Однако это трудный путь и требует высокой квалификации программиста. В остальных случаях построение буферного пула выполняет управляющая программа.

При втором способе статическое построение буферного пула по макрокоманде BUILD производится, если известны число буферов и их размер. Формат макрокоманды

[метка] BUILD адрес области, {число буферов, длина буфера}

где адрес области задает адрес памяти для построения буферного пула; число буферов и длина буфера определяются либо непосредственно, либо через регистр "0". Буферный пул можно создавать для нескольких НД, при этом число буферов в пуле будет равно суммарному числу буферов для каждого набора. Рассмотрим пример построения буферного пула для двух НД:

```
          . . . . .
          BUILD   POOL, 6, 100
          . . . . .
          OPEN    (IN1,,OUT1, (OUTPUT))
          . . . . .
END       CLOSE   (IN1 ,, OUT1)
          . . . . .
          RETURN
IN1       DCB     DSORG=PS,  MACRF=(CL),BUFNO=2,BUFCB=POOL,EODAD=END
OUT1      DCB     DSORG=PS,MACRF=(PM),BUFNO=4,BUFCB=POOL
POOL      DS      CL 610
```

Макрокоманда организует буферный пул из 6 буферов, каждый по 100 байт. Два буфера отводятся набору данных IN1, четыре – OUT1. Оба набо-

ра совместно используют буферный пул POOL , как указано в макрокоманде DCB.

Третий способ построения буферного пула динамический и осуществляется, когда заранее не известно число и размер буферов, которые определяются в программе в процессе выполнения. Для этого используется МК GETPOOL, которая сама осуществляет запрос на требуемый объем памяти. При этом буферный пул может быть использован для одного набора данных. Формат макрокоманды

[метка] GETPOOL адрес DCB , { число буферов, длина буфера }

где задается адрес блока DCB , которому назначается буферный пул.

Освобождение буферного пула производится по МК FREEPOOL:

[метка] FREEPOOL адрес DCB

Четвертый способ построения буферного пула автоматический и осуществляется во время открытия НД.Это самый простой способ,так как нужно указать в блоке DCB (с помощью МК DCB) число и размер буферов. Память, выделяемая для буферного пула, освобождается после закрытия НД и выполнения МК FREEPOOL.

Другие макрокоманды. Для работы с библиотечными НД используются МК BLDL, FIND, STOW . Макрокоманда BLDL пересылает информацию о разделах библиотеки в справочный список, расположенный в памяти. Начальный адрес заданного раздела библиотеки определяется по МК FIND и помещается в блок DCB. Макрокоманда STOW вносит изменения в справочник библиотеки путем добавления, изменения, замещения или исключения одного элемента.

Для обработки ошибок ввода-вывода используются МК SYNADAF , SYNADRLS. Макрокоманда SYNADAF предназначена для анализа ошибок ввода-вывода и подготовки сообщения об этой ошибке. В основной памяти по этой МК организуется буфер сообщений в формате логической записи переменной длины. Адрес буфера помещается в регистр 1. За этим буфером строится своя область сохранения. Обычно МК SYNADAF используется в программе анализа ошибок SYNAD.

Перед выходом из программы анализа ошибок SYNAD необходимо восстановить регистр 13 (адрес области сохранения) , поместив туда адрес старой области сохранения, а также регистры 2–12, если они указаны в программе. Для этого необходимо освободить память, выделенную под буфер сообщения и новую область сохранения, выдав макрокоманду SYNADRLS. Перед этим в регистр 13 должен быть помещен адрес области сохранения, созданной макрокомандой SYNADAF .

Макрокоманды виртуального метода доступа. Для работы с НД VSAM использует МК построения управляющих блоков и обработки запроса.

Перед началом работы с НД необходимо его открыть макрокомандой OPEN . При ее выполнении создаются управляющие блоки для обработки запроса на ввод-вывод и определяются программные средства для обработки НД. Для этого метод VSAM объединяет информацию из оператора DD с информацией из блоков управления доступом, списка выходов и характеристик НД, заданных в каталоге.

По окончании работы с НД для восстановления информации в блоках, ко-

торая была до открытия его, производится закрытие набора МК CLOSE. При этом освобождается использованная память.

Макрокоманда ACB строит блок управления доступом перед открытием НД для его обработки. Макрокоманда RPL создает блок управления запросом, который содержит параметры запроса. Макрокоманда EXLST строит список выходов на программы пользователя во время трансляции. Нужно задать выход на программы обработки логических ошибок, ввода-вывода, аппаратных сбоев, проверки разрешения доступа.

Макрокоманда GENCB применяется для построения блоков: управления доступом (ACB), построения списка выходов (EXLST), управления запросом (RFL) во время выполнения программы.

Для считывания записей используется МК GET, а для включения записей в НД – МК PUT. По МК POINT устанавливается начало обработки. Для перевода активной задачи в состояние ожидания до завершения обработки запроса и проверки условий завершения служит МК CHECK.

В методе доступа VSAM применяются два дополнительных оператора DD с именами JOBCAT и STEPCAT для определения каталогов VSAM. Описание НД VSAM может находиться в главном каталоге или каталоге пользователя.

Макрокоманды графического метода доступа. Программы графического метода доступа (ГМД) функционируют как часть управляющей программы ОС ЕС. Они используются для управления вводом-выводом с терминалами и обработки сигналов "Внимание". Макрокоманды ГМД служат для построения управляющих блоков и обращения к программам графического метода доступа. С помощью этих макрокоманд организуется связь пользователя с программами ГМД, а через них – с супервизором ввода-вывода.

ГМД дает возможность организовать режим диалога пользователя и программы через дисплей. В этом режиме возникают значительные перерывы в работе, во время которых пользователь анализирует результаты, обдумывает действия и т. д. На время таких перерывов программа пользователя переводится в режим ожидания, а ЭВМ выполняет программы других пользователей или решает фоновые задачи того же пользователя.

Макрокоманды, используемые в ГМД, можно условно разбить на три группы: ввода-вывода, базисного и специального методов обработки сигналов внимания. К МК ввода-вывода относятся: DCB – построить блок управления данными; OPEN и CLOSE – открыть и закрыть блок управления данными; GREAD – считать данные (передача данных от дисплея к ЭВМ); GWRITE – записать данные; GCNTRL – стереть данные в буфере дисплея. Макрокоманды базисного метода включают: SAES – определить программу внимания; SPAR и DAR – включить и выключить программу внимания. Макрокоманды специального метода предназначены для анализа ANALYS и обработки GSERV сигнала "Внимание". Подробно организация ввода-вывода описана в работе [25].

3.4. НАЗНАЧЕНИЕ И СТРУКТУРА БАЗ ДАННЫХ

Структуры данных для представления реального мира. Обработка цифровой информации о реальном мире требует представления данных в виде объек-

тов и связей между ними. Необходимо иметь ограниченные модели, которые рассматривали бы данные и связи между ними для конкретных прикладных проблем. Такая модель имеет два уровня: логический и физический. Первый задает логическую структуру данных, второй – преобразование этой структуры в физическое представление данных на внешней памяти и ее программную обработку.

Структура данных – это набор правил и ограничений, которые показывают связь между отдельными элементами данных. Для представления логических моделей реального мира используются следующие структуры данных: последовательные, иерархические (древовидные), табличные и ассоциативные.

В *последовательной структуре* устанавливается определенная связь между данным элементом и последующим. Примером являются списки по алфавиту. Физически такая структура может быть представлена в виде последовательной организации файла либо связного списка.

Древовидная (иерархическая) структура представляет собой перевернутое дерево, из корня и узлов которого исходят ветви, соответствующие связям. Выделяют два вида указателей: первый идентифицирует элемент на верхнем или нижнем уровне; указатели второго вида связывают элементы, расположенные в дереве на одном и том же уровне.

Более сложной структурой является сетевая, в которой в узел может входить более одной ветви. *Табличная структура* задает связь между атрибутами на основе обычной таблицы. *Ассоциативная структура* устанавливает связь между единицами данных, имеющими общее свойство. Так, свойство "Студент" в ведомости будет иметь одно и то же значение для всех студентов данной группы, например "Сдаваемый предмет".

На практике наибольшее распространение получили три структуры данных: табличная, древовидная, сетевая, а также их комбинации. Однако существующие языки программирования не обеспечивают средств обработки всех этих структур.

Понятие о базе данных. Традиционная файловая организация данных имеет ряд недостатков: избыточность (одни и те же данные могут храниться в различных файлах), зависимость программ от структуры обрабатываемых данных, трудность обеспечения достоверности данных и их защиты, трудности расширений и т. д. Все это привело к тому, что в начале 70-х годов сформировалось понятие базы данных.

Под *базой данных* (БД) понимается именованная совокупность данных, отображающая состояние объекта и их отношений в рассматриваемой предметной области. Для работы с БД вводится комплекс языковых и программных средств, предназначенных для создания, ведения и использования данных базы, который называется *системой управления базой данных* (СУБД). База данных устраняет вышеуказанные недостатки файлов и, кроме того, обеспечивает ряд свойств. Прежде всего способы физического хранения данных могут изменяться без переделки прикладных программ. Добавление или расширение логических структур данных может производиться без их физического изменения. Для связи с программистом вводится интерфейс высокого уровня, что позволяет не учитывать сложности расположения данных и их адресации. Вводится язык запросов для получения требуемых данных без обращения к при-

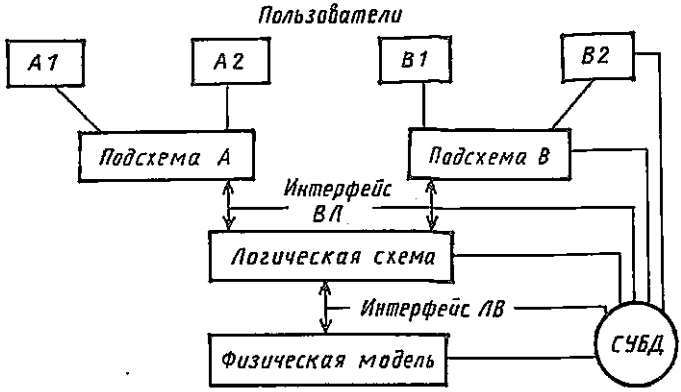


Рис. 3.8.
Архитектура базы данных.

кладным программам. Обеспечивается возможность автоматической реорганизации физической структуры данных.

Архитектура базы данных (ее многоуровневая организация) может быть разделена на три уровня: внешний, логический и внутренний (рис. 3.8). *Внешний уровень* соответствует представлению данных, которое понимается пользователем или программистом и описывается в прикладной программе. Описанию данных на внешнем уровне соответствует подсхема. *Логический уровень* определяет общую организацию данных в базе, на основании которых получаются различные внешние описания. Описанию данных на логическом уровне соответствует схема.

Внутренний уровень — это физическое представление данных и расположение их на внешней памяти. Оно зависит от используемых средств физического поиска данных: индикаторов, указателей, цепочек и т. д., определяется наличием области переполнения и средств включения и удаления записей.

На рис. 3.8 введены два вида отображения: между внешним и логическим уровнями (ВЛ), а также логическим и внутренним (ЛВ). Первое отображение определяет соответствие между подсхемами и схемой. Между этими двумя уровнями могут существовать отличия, например поля могут иметь различные типы данных, записи могут быть по-разному упорядочены и т. д. Так, одна подсхема может описывать списки студентов с указанием изучаемых ими предметов, а вторая — списки предметов с указанием фамилий студентов, которые их изучают.

Отображение ЛВ определяет соответствие между моделью данных и их физическим хранением. Если структура хранимых данных изменяется, то отображение ЛВ должно быть изменено так, чтобы логическая схема осталась неизменной.

Языковые средства БД. В БД введены понятия интерфейса пользователя и администратора. Интерфейс пользователя обеспечивается двумя видами языков: описания данных (ЯОД) и манипулирования данными (ЯМД). ЯОД включает средства для выполнения функций: идентифицировать элементы данных (поле, запись, файл); присваивать имена каждому типу элементов

данных; определять иерархию структур данных; определять связи между однотипными элементами. ЯОД может также определять тип кодирования данных (двоичный, символьный, десятичный), длину элементов данных, диапазон допустимых значений для элементов, количество элементов.

Для построения ЯОД могут использоваться либо расширения существующих языков программирования КОБОЛ, ПЛ/1, либо специальные языки. К последнему относится язык DL/1, который применяется как для описания логической базы данных, так и физической. Ассоциацией по языкам систем обработки данных (КОДАСИЛ) разработан другой ЯОД, который носит название КОДАСИЛ. Более подробно с этими ЯОД можно ознакомиться в работе [13].

Другая группа языков для работы с БД представляет языки манипулирования данными. Основной функцией ЯМД является выполнение операции при вводе-выводе данных. Различные СУБД включают разные наборы операторов ЯМД, но среди них можно выделить основные: ПОМЕСТИТЬ – позволяет включать в БД новые компоненты; УДАЛИТЬ – исключает из БД отдельные компоненты; ИЗМЕНИТЬ – позволяет корректировать содержимое БД; ИЗВЛЕЧЬ – из БД выбираются требуемые компоненты данных.

Практически во всех типовых СУБД реализуются два основных подхода построения ЯМД. Первый состоит в использовании в программах оператора CALL.

При втором подходе в набор операторов алгоритмического языка включается расширение для реализации функций ЯМД. В этом случае исходная программа сначала проходит этап предтрансляции, на которой операторы ЯМД заменяются группами операторов базового языка, после чего выполняется трансляция с базового языка (ПЛ/1, КОБОЛ).

Администратор БД – лицо (группа лиц), ответственное за общее функционирование БД. В его обязанности входит следующее: определение информационного содержания БД; определение структуры хранения и стратегии доступа; выполнение контроля полномочий и процедур проверки достоверности; определение стратегии дублирования и восстановления; реакция на изменение к требованиям.

Функционирование СУБД. Основные требования к СУБД сформулированы в предложениях группы КОДАСИЛ и заключаются в следующем: исполь-

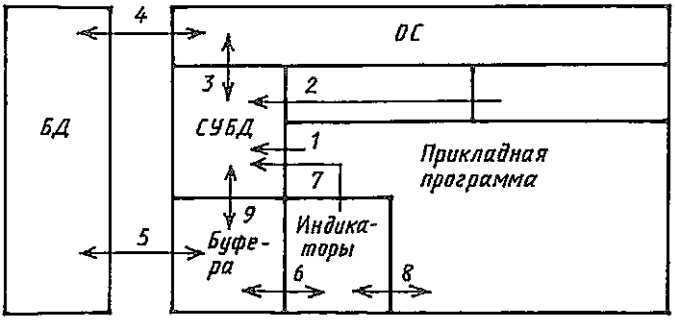


Рис. 3.9.
Схема работы СУБД.

зование структур данных для различных приложений без избыточности данных;описание данных независимо от логики программ; возможность описания табличных, древовидных и сетевых структур данных; обеспечение многообразия методов доступа; обеспечение защиты от несанкционированного доступа.

На рис. 3.9 показано взаимодействие прикладной программы (ПП), ОС и СУБД. При необходимости выполнения чтения или записи данных в ПП организуется запрос на ЯМД *1*. На основе анализа запроса СУБД добавляет к нему аргументы из описания схемы и подсхемы *2*, СУБД посылает запрос ОС на выполнение операции ввода-вывода *3*. ОС управляет операциями доступа к внешней памяти *4* и осуществляет передачу данных между БД и системными буферами *5*. Производится передача данных между внешней памятью и системным буфером с возможными преобразованиями *6*. СУБД сообщает ПП о выполненном запросе *7*. Обработка данных ПП осуществляется средствами языка программирования *8*. СУБД управляет работой системных буферов *9*.

3.5. ПЕРВИЧНАЯ ОБРАБОТКА ДАННЫХ

Основные понятия. Организация вычислительных процессов ЭВМ включает большое число подготовительных этапов, которые характеризуются низкой надежностью, но определяют достоверность последующих решений. Среди них важное значение имеет первичная обработка данных (ПОД), для повышения надежности которой вводят дополнительные программные средства. Прежде всего это простейшие средства контроля, программы, построенные с использованием генератора ввода, т. е. для каждого вида данных генерируется программа контроля на макроассемблере. В последнее время появилось много программ и пакетов для первичной обработки КОМПАКТ, САНБД, ВВОД КОР и т. д.

Принципы и средства ПОД. Рассмотрим построение и организацию одного из таких пакетов – банка первичных документов (БПД). В основу построения пакета БПД положены следующие принципы: исходные данные представляются в виде форматизированной структуры, состоящей из заголовка и строк; введены три уровня описания исходных данных – на внешнем, логическом и физическом уровнях, для которых существуют структурированные средства описания; контрольные соотношения исходных данных описываются непроцедурным языком контроля; исходные данные после ввода, контроля и корректировки могут поступать на обработку прикладными программами в БД общего назначения.

Рассмотрим формальное описание исходных данных иерархической структуры, включающей домен, строку, документ, файл соответственно снизу вверх.

Опишем файл в виде $F = \left\{ D_j^1 \mid j = 1, \overline{n_1} \mid, D_j^2 \mid j = 1, \overline{n_2} \mid, \ldots, D_j^k \mid j = 1, \overline{c_k} \mid \right\}$, где k – количество разнотипных документов.

Определим документ как

$$D = \left\{ C_i^1 \mid i = \overline{1, k_1} \mid, C_i^2 \mid i = \overline{1, k_2} \mid, \ldots, C_i^t \mid i = \overline{1, k_t} \mid \right\},$$

где t – количество разнотипных строк.

Строку представим в виде

$$C=\left\{d_l^1 \mid l=\overline{1,m_1} \mid,\ d_l^2 \mid l=\overline{1,m_2} \mid, \ldots, d_l^p \mid l=\overline{1,^{l}mp} \mid\right\},$$

где d – количество различных доменов.

Наконец, домен запишем в виде

$$d=\left\{a_r^1 \mid r=\overline{1,u_1} \mid, a_r^2 \mid r=\overline{1,u_2} \mid, \ldots, a_r^s \mid r=\overline{1,us} \mid\right\},$$

где a – элемент алфавита исходных символов; s – его мощность.

На логическом уровне для представления всех типов данных введем универсальное описание $U=(Z, C_I, C_o S_s, O_p)$, где Z – заголовок; C_I – список идентификаторов; C_o – адресные ссылки описаний; S_s – адресные ссылки связей; O_p – описание параметров.

Такое представление исходных данных позволяет однообразно описывать их на внешнем уровне и обрабатывать различные классы данных на логическом. Для описания на внешнем уровне язык включает следующие операторы:

ДОМЕН: имя домена ⟵ параметры;

СТРОКА: имя строки ⟵ входящие домены ⟵ параметры;

ДОКУМЕНТ: имя документа ⟵ входящие строки ⟵ параметры;

СВЯЗЬ: имя связи ⟵ имя связанных объектов данных.

Наличие такого ЯОД позволяет легко описать файл сложной иерархической структуры со связями между различными элементами.

Язык контроля документов. Для контроля соотношений между элементами иерархических данных в файле пишется программа на соответствующем языке. Контроль производится на уровне домена строки и документа и записывается набором формул контроля. Результат контроля находит отражение в виде распечатки документа, в которой помечаются как ошибочные (*) значения доменов, строк или межстроковых соотношений, не удовлетворяющих формулам контроля.

Контроль документов осуществляется на уровне заголовка документа, межстроковых соотношений и доменов строки. Для этого программа контроля разбивается на блоки. Пусть заголовок $Z=(K, P)$, где K – ключевые значения; P – параметрические. Уровень заголовка служит для контроля параметрических и ключевых доменов. На уровне строки организуется циклический просмотр строк на соответствие записываемым выражениям. На уровне документа проверяются контрольные суммы с использованием встроенных функций: сумма по вертикали, количество информационных строк и т. д. Предусмотрен контроль на дублирование строк, сравнение со справочником на всех уровнях контроля. Для домена применяется контроль по модулю 11. Для проверки межстроковых соотношений введена функция дублирования (повтор одинаковых строк – ошибка). Программа контроля начинается оператором PROC и заканчивается оператором END.

Для реализации вышеизложенного структура пакета ПОД интерпретирующего типа включает программное, языковое и информационное обеспечение. На входе документы описываются в виде макета и отражают структуру и форму документов на носителе. Проверка достоверности документов осуществляется по формулам контроля, записанным на ЯКД для логического описания документов (независимо от носителей). Для этого макет преобразуется и за-

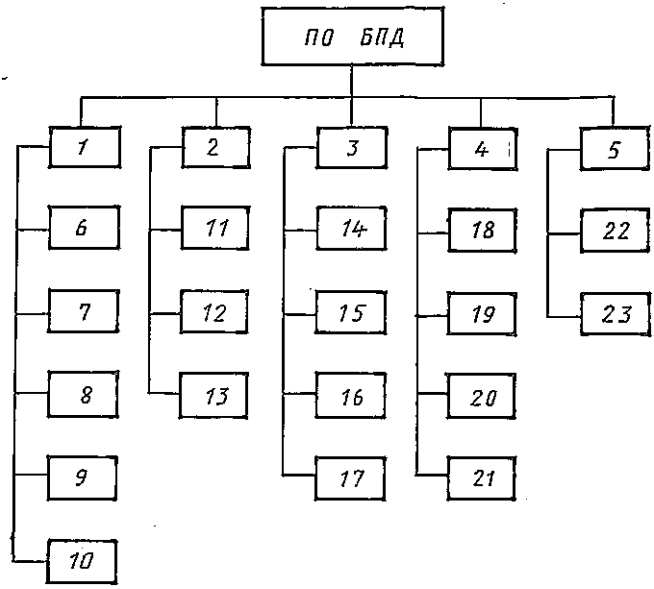


Рис. 3.10.
Структура ПО банка первичных документов.

писывается в базу в виде схемы. После контроля ошибочные документы поступают на коррекцию. Из базы документ выбирается по структуре шаблонов, которых, как и макетов, может быть несколько.

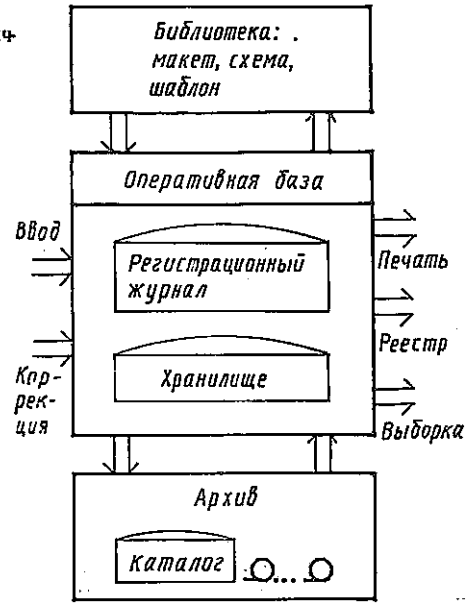
Программное обеспечение БПД делится на ряд подсистем (рис. 3.10). Главная подсистема *1* реализует основные функции БПД: ввод документов в базу *6* , контроль достоверности и распечатку протоколов ошибок *7* , корректировку, контроль и распечатку откорректированных документов *8* , распечатку списка документов по заданной форме *9* , выборку документов из базы по простому или сложному запросу *10* .

Подсистема создания описания функций *2* включает создание описания схем на соответствующем входном языке, а также макетов и шаблонов документов *11* , трансляцию их во внутреннее табличное представление *12* , построение интерпретатора на основании схемы и программы контроля.

Подсистема инициализации *3* содержит функции создания служебных и рабочих НД для регистрационного журнала *14* , хранилища *15* , архива *16* и словаря *17* , описывающего данные на различных уровнях. Подсистема поддержки *4* включает следующие функции: получение справки о состоянии оперативной базы *18* , реорганизации базы *19* , копирование НД на ленту *20* , восстановление НД с ленты *21* . Подсистема архивизации *5* выполняет функции архивизации *22* и деархивизации *23* .

Лингвистическое и информационное обеспечение. В БПД используются следующие языки: описания данных, контроля данных, запросов, корректировки и управления. Основные конструкции первых двух языков описаны выше. При реализации ЯОД удалось обойтись без лексического анализатора,

Рис. 3.11.
Информационное обеспечение банка первичных документов.



поскольку для этой цели применялась конструкция GET DATA языка ПЛ/1. Язык запросов используется для задания определенного множества документов, имеющихся в базе, в операциях выборки и взятия реестра. Основу такого языка представляет выражение: имя домена ⌙ предикат значения. Несколько таких выражений, записанных подряд, считаются связанными операцией конъюнкции и задают сложный запрос. В БПД применяется язык корректировки табличного типа. Графы бланка корректур определяют место расположения ключевых слов и значений. Синтаксис корректуры задается кодом корректировки. Язык управления отражает порядок выполнения функции БПД для определенного технологического процесса. Для этих целей используется язык управления заданий ОС, причем технологический цикл обработки оформляется в виде процедур, собранных в библиотеку.

Схема информационного обеспечения БПД (рис. 3.11) включает оперативную базу, состоящую из хранилища, каталога, архива и библиотек инструкций. База служит для временного хранения документов на период обработки, причем каталог имеет индексно-последовательную, а хранилище — прямую организацию. Архив включает документы долговременного хранения, имеет последовательную организацию.

3.6. ПОСТРОЕНИЕ СИСТЕМЫ УПРАВЛЕНИЯ ДАННЫМИ

Подход. Рассмотрим построения БД и СУБД для работы с последовательными наборами, включающими массивы и переменные. Такие данные используются для работы с пакетами программ научных расчетов, логического проектирования, обработки экспериментов. Использование стандартных СУБД типа ОКА, ИНЕС, СЕТОР и аналогичных нецелесообразно из-за их большого объ-

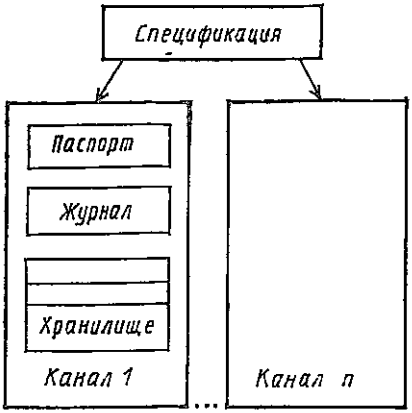


Рис. 3.12.
Структура специализированной БД.

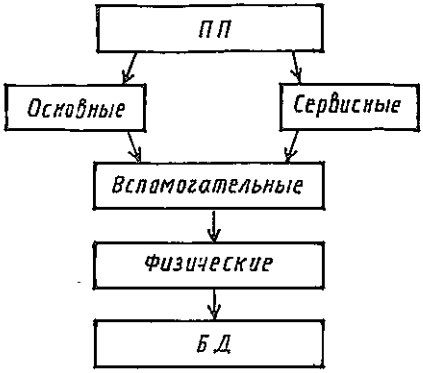


Рис. 3.13.
Структура специализированной СУБД.

ема, сложности языковых средств, больших сроков привязки к пакетам ПП и трудности освоения.

Построение БД. Построение БД включает анализ данных, преобразование данных в структуры хранения (логический этап), определение этих структур и методов доступа (физический этап). Рассмотрим подробнее второй и третий этапы. Поставим в соответствие каждому этапу пакета прикладных программ (ППП) понятие канала. Под последним будем понимать служебную и проблемную информацию, связанную с этапом. Описание служебной и проблемной информации по этапам сведем в НД "Спецификация". В нем будут содержаться сведения об именах НД каналов и их длине.

Под каналом понимается тройка $K = (P, J, H)$, где P – паспорт; J – журнал; H – хранилище (рис. 3.12). В хранилище помещаются порции информации (запись, группа записей, массивы) по этапу расчета, сопровождаемые идентифицирующими данными. Последние в виде ключевых доменов записываются в журнал, каждая строка которого характеризует порцию данных хранилища. Паспорт используется для описания структуры журнала и задания предметных имен каждому его полю. Для введенных понятий выбраны следующие организации данных. Спецификация БД представляет библиотечный набор, в котором каждому каналу соответствует свой раздел по имени. В разделе описываются характеристики канала, например имя журнала, паспорта и хранилища, их организация НД, длина записи, блока и т. д.

Паспорт является также библиотечным набором, в котором перечислены ключевые слова предметной области и их длина, например: отдел 8 символов, фамилия 15 символов и т. д. Журнал представляет индексно-последовательный набор, каждое поле которого определяет ключевые слова паспорта, например: 323 (отдел), Петров (фамилия) и т. д. Кроме ключевых полей, журнал включает системные поля: пароль, дату, номер, физический адрес в хранилище.

На физическом уровне хранилище представляет библиотеку, длина записи и блока которой определяется в описании канала. Каждой порции информации из ПП соответствует раздел. Доступ к разделу может осуществляться одним

из последовательных методов QSAM или BSAM. Выбор метода можно задавать в спецификации. В процессе работы порция информации из ПП с заданными ключевыми элементами будет записываться в базу либо читаться из нее.

Подсистема администратора выполняет функции создания БД и поддержания ее в рабочем состоянии. Для рассматриваемого примера функции следующие: создание и заполнение библиотеки спецификации, создание канала, реорганизация канала. При создании канала необходимо добавить в библиотеку спецификаций раздел с описанием нового канала, создать паспорт, а также пару журнал–хранилище. Реорганизация может заключаться в изменении паспорта и журнала. Для выполнения этих функций в распоряжении администратора должна быть библиотека процедур, которые выполняются стандартными средствами.

Построение СУБД. Основными функциями БД при работе с пакетом программ являются чтение и запись порций информации. ПП после выбора канала обращается к СУБД для записи данных с их регистрацией в терминах предметной области, определяемых в паспорте. СУБД выделяет программе раздел хранилища и присваивает ему имя – регистрационный номер. Далее выполняется запись информации в данный раздел хранилища (виртуальный последовательный файл) средствами СУБД. При этом обеспечивается создание DD-предложения, открытие и закрытие набора. По завершении непосредственно записи производится регистрация записанной информации в журнале. Запись в журнале устанавливает связь между характеристиками записанной информации и регистрационным номером.

Для получения порции информации программа после выбора канала по имени обращается к СУБД с запросом в терминах той же предметной области. СУБД на основании запроса находит запись в журнале, а по ней – раздел хранилища, обеспечивает создание DD-предложения, открытие, чтение и закрытие виртуального файла. Для обеспечения вышеперечисленных функций структура СУБД включает подсистемы: основную, сервисную, вспомогательную и физическую (рис. 3.13).

Основная подсистема включает программы открытия канала, чтения из хранилища, записи в хранилище, закрытия канала. Сервисная подсистема обеспечивает функции чтения по регистрационному номеру, удаление из базы по ключу или регистрационному номеру. Вспомогательные программы выполняют функции: формирования ключей и записей в журнале, формирования таблицы абстрактных данных по записи журнала, занесения и взятия ключа по этой таблице, заполнения регистрационной таблицы при запросе по неполному ключу. Эти программы реализуют вспомогательные функции при чтении и записи и обеспечивают регистрацию основной информации в БД.

Программы на физическом уровне выполняют чтение, запись, удаление раздела хранилища, создание DD-карты для доступа к разделу, определение физического адреса для вновь созданного раздела, определение раздела с заданным именем. Кроме того, для работы с базой на нескольких терминалах необходимы программы захвата канала и его освобождения как ресурса.

Привязка БД к пакету программ. Привязка выполняется следующим образом: 1) в соответствии с этапами работы пакета программ выбирается число каналов и производится их описание; 2) для регистрирующей информации в терминах предметной области составляются пары: абстрактное имя ключа –

значение (фамилия–Петров); 3) заносятся абстрактные имена ключей и их максимальная длина в паспорт; 4) в соответствующих местах ПП производится обращение к БД, как и при работе с виртуальным файлом по имени DAMMY RECORD. Перед обращением канал БД открывается, а после завершения операции закрывается.

При работе с ПП средствами языка ПЛ/1 структура обращения при чтении следующая:

```
DCL ... % INCLUDE DATA          /* переменные для связи с СУБД*/
...
% INCLUDE OPEN                  /* открытие канала БД*/
...
% INCLUDE PUTNAME               /* заполнение регистрационной таблицы*/
% INCLUDE READ                  /*чтение порции информации*/
% INCLUDE ANALIZ                /* анализ кода возврата */
% INCLUDE CLOSE                 /* закрытие канала БД*/
```

Каждому оператору INCLUDE соответствует предпроцессорная секция, которая обрабатывается во время трансляции. При записи информации аналогично описываются переменные для связи с БД, производится открытие канала, формирование ключей записи, непосредственно запись в раздел хранилища из ПП, проверка условий записи и закрытие БД. При реализации вышерассмотренной СУБД подсистемы основная, вспомогательная и сервисная реализуются на языке высокого уровня (ПЛ/1, ПАСКАЛЬ), а программы физического уровня – на ассемблере, что связано с обращением в них к МК управления данными.

Использование описанной СУБД удобно для работы с обычными библиотеками в терминах предметной области. Для этого они помещаются в хранилища под любыми удобными для пользователя наборами предметных ключей. Другое применение СУБД связано с помещением в хранилище под предметными именами (ключами) пусковых пакетов типовых заданий. В обоих случаях такое использование СУБД можно рассматривать как функционирование на ЕС ЭВМ так называемой электронной записной книжки.

Выводы

1. На устройствах внешней памяти существуют четыре основные организации наборов данных (по расположению записей) : последовательная, индексно-последовательная, прямая и библиотечная.

2. На физическом уровне ввода-вывода программист может создавать свою программу ввода-вывода, используя метод доступа EXCP. На логическом уровне программист может применять программы 6 основных методов доступа: QSAM, QISAM, BSAM, BISAM, BPAM, BDAM. Кроме того, для работы с виртуальной памятью программист использует виртуальный метод доступа, а для поддержания средств машинной графики – графический метод доступа.

3. Организация вычислительных процессов при работе с наборами данных на уровне программиста реализуется рядом макрокоманд. К основным относятся: МК открытия и закрытия наборов (OPEN, CLOSE), чтения и записи (GET, PUT и READ, WRITE). Описание наборов данных производится МК DCB. Построение буферного пула осуществляется

МК BUILD и GETPOOL, освобождение – FREEPOOL. Виртуальный и графический методы доступа используют дополнительные МК.

4. Для совместной работы нескольких пользователей с большим объемом данных реализуется их централизованное хранение в базе данных, с которой работает ряд специальных программ, объединенных в СУБД. Для представления данных в базе служит язык описания данных, а для работы с данными – язык манипулирования данными. На физическом уровне СУБД использует программы ОС.

5. При работе с большим объемом логически связанных данных необходима их первичная обработка (описание логических связей, контроль, корректировка). Для этих целей используют специальные пакеты, включающие набор языковых средств для описания данных на различных уровнях, контроля данных и их обработки. Пакет может содержать набор программных и информационных средств и представляет собой специализированную СУБД.

6. Для построения специализированной БД типа "Электронная записная книжка" в среде ОС ЕС организуется тройка: паспорт, журнал, хранилище. В последнее под предметными именами, хранимыми в журкале, заносится произвольная порция информации. Описание полей журнала задается в паспорте. Для работы с данными такой БД на языке ПЛ/1 пишется ряд логических программ, а на физическом уровне – ассемблерных программ, входящих в СУБД.

Рекомендуемая литература: [2, 8, 19, 13, 25].

Вопросы для контроля

1. Поясните понятие метода доступа, организации данных.
2. В чем сущность прямой и библиотечной организации?
3. Определите функциональное назначение блока DCB.
4. Объясните уровень логического ввода-вывода.
5. Приведите пример использования макрокоманд GET, PUT.
6. Приведите пример использования макрокоманд WRITE, READ.
7. Поясните назначение МК OPEN, CLOSE.
8. В чем заключается организация работы с буферами?
9. Поясните отличие базы данных от работы с файлами.
10. Сравните логический и физический уровни БД.
11. В чем назначение первичной обработки данных?

4. РЕЖИМ РАЗДЕЛЕНИЯ ВРЕМЕНИ И ТЕЛЕОБРАБОТКА ДАННЫХ

4.1. СОСТАВ И ФУНКЦИИ СИСТЕМЫ С РАЗДЕЛЕНИЕМ ВРЕМЕНИ

Общие сведения. Система разделения времени (СРВ) представляет собой комплекс языковых и программных средств для расширения возможности ОС за счет диалоговой работы нескольких пользователей. В качестве средства доступа к системе пользователи СРВ применяют диалоговые комплексы (ЕС-7906, ЕС-7920 для ЕС ЭВМ). В режиме СРВ все ресурсы системы делятся между высокоприоритетными заданиями разделения (РВ) и низкоприоритетными заданиями пакетной обработки. Один раздел ОС отводится для управляющих программ СРВ, другие – под задания пользователей (разделы РВ). Несколько заданий могут разделять один раздел, образуя одну или несколько очередей, причем в течение промежутка времени (кванта) там находится одно задание, остальные – во внешней памяти. Порядок выполнения заданий определяется управляющими программами СРВ и диспетчером ОС. По истечении кванта времени задания образ задания (содержимое раздела ОП) записывается во внешнюю память, а в свободный раздел вводится задание, которому выделяется квант времени. Процесс пересылки задания из ОП во внешнюю память называется вытеснением, а обратно – восстановлением.

Связь абонента с системой осуществляется посредством языка команд. *Сеансом* называется время от начала работы по команде LOGON до конца работы по команде LOGOFF. В начале работы система назначает заданию раздел памяти и ресурсы, по команде LOGOFF они освобождаются. Для поддержания диалога с пользователем система выдает набор сообщений, которые информируют о готовности принять команду (READY), необходимости ввода данных, наличии ошибок ввода команд или трансляции, программ, возможных действиях. Абоненту необязательно помнить форматы команд, поскольку по команде HELP можно получить информацию обо всех возможностях системы, любой команде системы, ее функциях, синтаксисе и т. д.

Состав СРВ. Система включает управляющую программу, программы обработки команд языка СРВ, монитор и служебные программы. Управляющая программа выполняет функции по проверке санкций абонента СРВ, оформляет задания, выделяет для них кванты времени, осуществляет смену заданий РВ и обеспечивает связь между абонентами. Эти функции реализуют следующие прграммы: управления РВ, управления разделом, планировщик абонента, координатор РВ, диспетчер РВ, координатор ввода-вывода. Процесс управления в СРВ условно разбивается на пять уровней (рис. 4.1).

Рассмотрим их назначение.

1. Программа управления РВ инициирует режимы работы РВ, изменяет характеристики системы, производит обмен заданий РВ, обслуживает входную и выходную очереди запросов на обмен. По команде START вызывается модуль инициализации задачи, который определяет размер памяти и получает от ОС

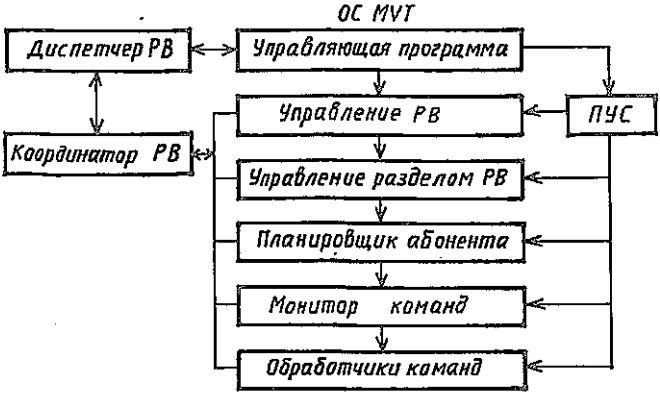


Рис. 4.1.
Иерархические уровни управления в СРВ.

раздел, в котором для задачи строятся управляющие блоки и буфера. Затем вызывается программа управления разделом для каждого раздела РВ.

2. Программа управления разделом контролирует завершение заданий и их восстановление, находится в разделе управления СРВ, является реентерабельной. Для каждого раздела создается своя задача управления разделом, которая контролирует восстановление и завершение заданий. Раздел РВ содержит область локальных очередей, в которой хранятся управляющие блоки. При вытеснении задания блок тоже вытесняется, а затем восстанавливается.

3. Планировщик абонента вызывается программой управления разделом при начале и завершении сеанса. Он выполняет функции системного ввода и инициатора для задания РВ. Планировщик и все программы ниже его подвергаются обмену, поскольку находятся в разделе РВ. По некоторым операндам команды LOGON строятся операторы JOB и EXEC для задания РВ. В операторе EXEC задается имя выполняемой процедуры, которая включает имя программы и операторы DD для наборов данных абонента. Операторы JOB и EXEC передаются планировщику для определения требуемых ресурсов, а затем инициатору для их распределения и вызова выполняемой программы.

4. Планировщик абонента вызывает программу (монитор), указанную в операторе EXEC процедуры LOGON . Монитор выполняет следующие функции: принимает команды; вызывает требуемые программы (обработчики команд) для выполнения команд пользователя; обрабатывает сигнал ВНИМАНИЕ; обрабатывает ошибки, возникающие при работе программы; возвращает управление планировщику абонента.

5. Обработчики команд находятся на нижнем уровне управления и выполняют действия, предусмотренные параметрами команд, а также организуют диалог для реализации подкоманд в команде.

Функционирование СРВ. Кроме вышеуказанных программ уровней, в системе имеется координатор РВ, который управляет распределением системных ресурсов. Он выделяет заданиям кванты времени на основании данных о размере программы, использованном времени процессора, объеме ввода-вывода для терминала, количестве готовых заданий.

Передачей параметров от команды START оператора или из системной библиотеки параметров можно управлять алгоритмами планирования с целью получения необходимой производительности для конкретной смеси заданий. Координатор используется программами всех уровней, которые сообщают ему о завершении кванта времени, абоненте, ожидающем сеансе, задании, вводе-выводе и т. д. На основании этого координатор составляет картину загрузки сисетмы, выполняет обмен, изменение приоритета, выделение абоненту раздела.

Диспетчер РВ, работающий на уровне управления ОС, получает управление от диспетчера ОС и реализует следующие функции: передает воздействия от координатора РВ (вытеснение или восстановление) управляющей программе РВ и программе управления разделом; управляет диспетчеризацией программ РВ и пакетных задач и переупорядочивает очередь активных задач; извещает координатор об изменении статуса старой или новой задачи.

Все сообщения от терминала или к нему обрабатываются координатором ввода-вывода, который выполняет связь между программами СРВ и телекоммуникационным методом доступа. Введенная команда абонента обрабатывается монитором, который вызывает обработчик команд. Для инициирования его работы монитор выдает макрокоманду ATTACH, а для завершения – DETACH и посылает сообщение READY на абонентский пункт (АП). При вводе новой команды прерванная программа снимается и монитор вызывает новую программу обработки.

Управление временем в СРВ. Для выполнения программы в СРВ выделяются кванты времени, при больших размерах которых увеличивается время ответа на запрос, а при малых затрачивается большой процент на обмен и выборку задач. Алгоритмы разделения времени в координаторе РВ определяются задаваемыми системными параметрами и информацией о сложившейся ситуации в процессе работы. Параметры координатора РВ задаются оператором DRIVER, в котором определяются количество циклов обслуживания для каждой очереди в разделе СРВ, процент использования процессора (процент времени СРВ от времени процессора, включая пакетный режим), число очередей и т. д.

Координатор при вычислении квантов требует два цикла. В первом все задания в очереди к разделу должны быть выбраны и восстановлены. Задается среднее время для обслуживания каждой очереди. Для вычисления главного кванта это время делится на число заданий в очереди, при этом может быть указана его минимальная величина. Очередям можно указать несколько циклов обслуживания, начиная с очереди с меньшим номером. В очереди с меньшими номерами помещают задания, требующие малой памяти или активного ввода-вывода, они обслуживаются чаще (квант меньше).

Время обмена между основной и внешней памятью зависит от объема выделяемой заданию основной памяти, которая может задаваться. Задания, объем требуемой памяти для которых превышает указанный в операторе DRIVER, перемещаются в очереди с большим номером. Там они будут обслуживаться реже, что уменьшит потери времени на ввод-вывод. Система размещает новое задание в очередь с минимальным числом заданий. Однако для варьирования раздела, в который помещается задание, может быть задан параметр, учитывающий среднее число заданий в нем для постоянной активности раздела.

Для пакетных заданий в операторе DRIVER можно указать процент времени процессора на их выполнение. Кроме главного кванта, находится вспомогательный во втором цикле работы координатора. Он вычисляется для разделов с изменяющимся числом заданий после истечения главного кванта.

Задачи на выполнение запускает диспетчер ОС, который просматривает очередь ТСВ, сформированную диспетчером СРВ на основании алгоритмов диспетчеризации. Для вычисления вспомогательного кванта $\Delta\tau_в$ используются три метода диспетчеризации. При *простом методе* $\Delta\tau_в$ равен доступному времени, оставшемуся до следующего планируемого вытеснения задания. Такой метод используется с одним разделом РВ. *Уравновешенный метод* применяется с несколькими разделами РВ, при этом $\Delta\tau_в$ равен доступному времени, деленному на число разделов. При *взвешенном методе* можно выделить дополнительное время заданиям, которые находятся в состоянии ввода-вывода.

4.2. ОРГАНИЗАЦИЯ РАБОТЫ АБОНЕНТА В СРВ

Идентификация и создание наборов данных. Для начала сеанса абонент СРВ должен включить терминал, ввести команду LOGON с указанием своего имени (при защите – пароль абонента), а также имени процедуры LOGON, которая описывает выделенные ему ресурсы. Имена и характеристики всех абонентов СРВ имеются в наборе данных SYS1.UADS, которые сравниваются с введенными. В случае отсутствия его характеристик абонент не сможет начать сеанс работы. Сеанс работы заканчивается по команде LOGOFF , после чего на терминал выводится сообщение об отключении от СРВ и при возможности счет за работу.

Абонент имеет возможность создавать, редактировать и сохранять наборы данных, в которых могут быть тексты программ, исходные данные, текстовая информация. Имена НД в СРВ составные, включающие поля: идентифицирующий уточнитель, имя, присвоенное абоненту, описательный уточнитель. При создании НД абонент указывает лишь второе поле, остальные формирует СРВ. Первое поле является идентификатором абонента из команды LOGON, третье определяет тип набора. Для библиотечных наборов абонент указывает имя раздела.

Создание и редактирование наборов данных осуществляется программой редактора, которая вызывается командой EDIT. Первым оператором в этой команде является имя набора, затем статус (NEW, OLD) – новый, старый. Далее определяется тип НД: программа на языке программирования (ASM, BASIC, COBOL, FORTG, PL1), текст (TEXT) или исходные данные (DATA).

В режиме NEW абонент вводит новые строки, для прекращения ввода выдается пустая строка и происходит переход к редактированию. В режиме редактирования имеется набор подкоманд вставки, изменения, удаления, просмотра, поиска и т. д. Для сохранения отредактированного текста служит подкоманда SAVE , а для выхода из редактора – подкоманда END. Если была создана программа, то она может использоваться как входная информация для соответствующего транслятора. Пример работы СРВ:

LOGON PETROV/PAROL⏎ ACCT (561212)	начало работы
READY	готовность СРВ принять команды
EDIT ST NEW FORTG	вызов редактора для создания программы

INPUT	приглашение ввода ФОРТРАН-программы
00010 DIMENSION A(10)	вводимый пользователем текст программы
. . . . .	
00100 END	конец программы
Пустая строка	завершение ввода
EDIT	выход в редактор
SAVE	сохранить НД
SAVED	НД сохранен
END	выход из редактора в систему

Редактор может работать с последовательными или библиотечными наборами, содержащими записи фиксированного или переменного формата. При создании набора данных строки его можно нумеровать операндом NUM. Редактирование непронумерованного НД можно производить указателем курсора. Для перемещения указателя используются подкоманды UP n (вверх на n строк), DOWN n (вниз на n строк), TOP, BOTTOM – перемещающие указатель в начало и конец НД. Подкоманда FIND ⌙ <цепочка> перемещает указатель на строку, содержащую заданную цепочку знаков.

Корректировка наборов данных. В команде EDIT корректировка производится рядом подкоманд: вставки и удаления n строк, ввода, изменения знаков. Подкоманда DELETE(D) удаляет одну или более строк из набора, после чего курсор устанавливается на строке, предшествующей удаляемой. Например, для удаления текущей строки выдается D* , десяти строк, начиная с текущей, – D*10.

Подкоманда INPUT(I) устанавливает режим ввода для создания наборов, добавления или замены данных в них. В подкоманде задается номер строки или текущая строка (*), с которой начинается ввод. В подкоманде может задаваться приращение для увеличения номера последующих строк, замены старых строк новыми или вставки новых без замены старых строк.

Подкоманда INSERT(IN) вставляет одну или более новых строк за строкой, отмеченной курсором. Подкоманда CHANGE(C) изменяет последовательность знаков в строках набора данных. В формате подкоманды задаются номера первой и последней изменяемых строк, цепочки изменяемых знаков и знаков, на которые она заменяется. Например, изменить последовательность знаков AA на BB в строках с 10-й по 20-ю: C 10 ⌙ 20/AA/BB

Для выполнения синтаксического анализа операторов программы используется подкоманда SCAN(SC). Например, SC*10 – провести синтаксический анализ десяти строк, начиная с текущей.

Имеется возможность перенумерации строк с помощью подкоманды RENUM , в которой указывается начало нумерации и приращения. Например, REN40 2 10 перенумеровывает НД, начиная с 10-й строки с шагом 2, номер новой строки начинается с 40-го.

Для отображения на экране терминала содержимого НД или его части служит подкоманда LIST(L), в которой задается диапазон вывода. Например, для вывода текста с 10-й по 20-ю строки необходимо ввести строку L 10 20.

Для выхода из редактора используется подкоманда END.

Другие команды СРВ. Команда RENAME (REN) изменяет имя каталогизированного набора данных или раздела библиотеки. В команде указывается

старое и новое имя. Например, REN TEXT1 TEXT2 изменяет имя НД TEXT1 на TEXT2.

Для удаления НД используется команда DELETE(D), в которой указываются имена удаляемых наборов. По команде D (TEXT1, TEXT2) удаляются два набора с именами TEXT1 и TEXT2. Для изменения характеристик НД служит команда ATTRIB , операнды которой соответствуют операндам макрокоманды DCB. Например, ATTRIB TEXT1, LRECL(80) BLKSIZE(320), RECFM (FB) задает НД с именем TEXT1, длиной записи 80 байт, блока – 320 байт, организации – фиксированной, блокированной.

Команда ALLOCATE динамически распределяет НД для выполнения программы. По команде FREE происходит динамическое освобождение распределенных НД, изменяется выходной класс набора SYSOUT и удаляется список характеристик, определенных абонентом.

Информацию о распределенных для абонента или каталогизированных НД можно получить с помощью команд LISTALC, LISTCAT, LISTDS. Первая выводит на терминал список всех НД, назначенных абоненту, вторая команда – на терминал имена всех каталогизированных НД с идентификатором абонента, указанного в команде LOGON. Третья команда позволяет получить следующие характеристики НД: регистрационный номер тома, формат и длину логической записи, размер блока и т. д.

4.3. ТРАНСЛЯЦИЯ, РЕДАКТИРОВАНИЕ СВЯЗЕЙ, ОТЛАДКА И ВЫПОЛНЕНИЕ ПРОГРАММ

Работа с транслятором. Для вызова транслятора с терминала существуют три способа: с помощью специальных команд, относящихся к трансляторам; команды RUN; подкоманды RUN редактора EDIT.

Специальные команды: ASM – ассемблер, FORTSE – ФОРТРАН, PL1 – ПЛ/1, COBOL – КОБОЛ предназначены для выполнения трансляции программы, написанной на соответствующем языке.

Трансляция, загрузка и выполнение исходной программы осуществляются командой RUN или подкомандой RUN редактора. В команде RUN указывается имя НД, содержащего исходную программу, и язык программирования. В подкоманде RUN команды EDIT не задаются исходная программа и язык программирования, поскольку они определяются при работе с редактором. Например, для трансляции программы, созданной в редакторе EDIT , необходимо выдать подкоманду: RUN *MU AST* , в которой записаны передаваемые программе параметры.

Редактирование связей. Для редактирования связей служит команда LINK, в которой указываются имена НД с программами, подключаемые редактором. Для задания имени раздела библиотеки, в которой помещается загрузочный модуль, используется операнд LOAD. При его отсутствии система РВ присвоит загрузочному модулю имя объектного модуля. Например, необходимо отредактировать модуль из НД ANY и поместить в раздел T1 библиотеки BIB1. Для этого выдается: LINK ANY LOAD (BIB1 (T1)) . При поиске объектных или загрузочных модулей в библиотеках ФОРТРАНа, ПЛ/1 для разрешения внешних ссылок в команде LINK необходимо указать соответственно операнды FORTLIB, PL1LIB . В операнде LIB можно задать имена библиотечных набо-

ров, которые будут просматриваться редактором связей. Например, для редактирования объектного модуля из НД ANY и помещения загрузочного модуля в раздел TEMPNAME набора STU, а также разрешения внешних ссылок из НД SYS1.PLILIB(ANY) необходимо выдать последовательность LINK ANY PLILIB LIB(AN1) LOAD(STU).

Отладка программы. В СРВ включены средства для отладки программ, написанных на языках программирования ассемблер, ФОРТРАН, ПЛ/1, КОБОЛ. Эти средства позволяют останавливать программу в любой точке, выводить на терминал содержимое основной памяти, устанавливать точки прерывания и продолжать программу с любой точки.

Для отладки служит команда TEST: TEST ⌙ имя программы ⌙ ['параметры'] [$\begin{matrix}\text{LOAD}\\\text{OBJECT}\end{matrix}$]. Отлаживаемая программа находится в НД, указанном по имени, и может быть представлена в виде объектного (OBJECT) или загрузочного (LOAD) модуля. При остановке в некоторой точке выполняемой программы можно изменить содержимое ячеек памяти или регистров. Для этого используется команда ПРИСВОИТЬ ЗНАЧЕНИЕ с форматом: адрес ⌙ тип данных ⌙ 'данные'. Адрес может быть задан в символьной, абсолютной, относительной формах или в регистре. Тип данных может принимать значения: C – знаковые; H и F – двоичные с фиксированной точкой (полуслово и слово); E и D – с плавающей точкой; X – шестнадцатеричные; B – двоичные и т. д. Например, 1000 X '10' меняет содержимое ячейки 1000 на 10.

Команда TEST имеет ряд подкоманд для реализации функций отладки. Подкоманда AT используется для указания точек прерывания программы, в которых абонент будет производить нужные ему действия. Подкоманда CALL позволяет выполнить программу с указанного адреса, при этом указывается адрес возврата. Подкоманда RUN вызывает выполнение тестируемой программы с указанного адреса, игнорируя точки прерывания, и завершает выполнение команды TEST.

Для пересылки содержимого областей памяти (регистров) либо регистров в память и наоборот используется подкоманда COPY(C). В ней указываются адреса исходной и конечной областей, а также длина исходной области. Например, C 100 1000 500 производит пересылку 500 байт с адреса 100 в область 1000.

Для выполнения программы с указанного адреса используется подкоманда GO. При отсутствии адреса выполнение программы начинается с адреса последнего прерывания. Например, GO 500 означает выполнение программы с адреса 500.

Подкоманда LOAD(L) загружает отлаживаемую программу в основную память, а подкоманда DELETE(D) удаляет ее из памяти. Например, L A1 – загрузить, а D A1 – удалить модуль A1.

Для выделения запрошенного числа байтов памяти используется подкоманда GETMAIN с возможным указанием символического имени полученной области памяти. Для освобождения этой области памяти служит подкоманда FREEMAIN.

Подкоманда LIST выводит на терминал или в указанный набор данных содержимое области памяти или регистров. Область памяти можно задать на-

чальным и конечным адресом либо списком, если этих областей несколько. Например, L 100 200 – вывести содержимое памяти с адреса 100 по 200 адрес.

Имеются возможности просматривать системные данные. Так, подкоманда LISTDCB выводит на терминал или в НД содержимое блока управления данными. Подкоманда LISTPSW позволяет просматривать содержимое слова состояния программы. Подкоманда LISTTCB выводит содержимое блока управления задачей TCB. Для вывода схемы распределения основной памяти, выделенной программе, используется подкоманда LISTMAP, для прекращения работы программы TEST – подкоманда END.

Выполнение программы. Для выполнения отредактированной (отлаженной) программы служит команда CALL. Перед ее выдачей необходимо определить все требуемые НД. В команде CALL задается имя библиотеки и раздела, где находится загрузочный модуль. Например, CALL ANY (TI). Если программа находится в разделе TERMINATE НД ANY , ее можно выполнить по команде CALL ANY. В команде CALL можно указать параметры для выполняемой программы CALL ANY 'PAR1, PAR2'.

Обработка пакетных заданий. С терминала в СРВ можно производить выполнение пакетных заданий с отображением результатов на печати либо экране. Такое задание должно состоять из операторов ЯУЗ и может включать текст программ и данных. Задание может создаваться редактором текста EDIT в виде последовательного набора или раздела библиотеки. При отсутствии оператора JOB в пакетном задании он генерируется системой.

Для управления пакетными заданиями имеется ряд команд, которые могут использовать абоненты. Для передачи на обработку одного или нескольких заданий служит команда SUBMIT. Например, SUB A1 запускает на обработку задание A1.

Команда STATUS выводит на терминал состояние задания, указанного по имени. Команда OUTPUT позволяет направить результаты выполнения задания на экран либо в указанный НД, а также изменяет или удаляет класс выходных наборов. Подкоманда SAVE переименовывает и каталогизирует НД. Например, SAVE A1 переименовывает старый набор в набор с именем A1. Команда CANCEL(C) прекращает выполнение заданий, список имен которых указывается в команде. Например, CANCEL A1 прекращает выполнение задания A1.

4.4. СТРУКТУРА И ФУНКЦИОНИРОВАНИЕ СИСТЕМЫ ТЕЛЕОБРАБОТКИ ДАННЫХ

Состав. *Телеобработкой данных* называется обработка данных в ВС, пользователи которой удалены от ЭВМ и применяют средства дистанционной передачи данных. Система телеобработки данных (СТД) представляет собой комплекс технических и программных средств, обеспечивающих пользователям удаленных станций доступ к ресурсам ВС.

Разработанный в странах – членах СЭВ единый комплекс технических и программных средств телеобработки данных обеспечивает реализацию диалоговой и пакетной обработки данных, их сбор и передачу, дистанционную работу с базами данных, а также обмен информацией между различными ЭВМ в режиме сетевой обработки.

СТД и сети представляют собой совокупность аппаратурных и программных средств передачи и обработки данных. Аппаратурные средства включают ЭВМ, мультиплексоры передачи данных (МПД), групповые устройства обработки, аппаратуру передачи данных (АПД), каналы связи, АП (рис. 4.2).

Программные средства СТД включают: базисный телекоммутационный метод доступа (БТМД), общий телекоммутационный метод доступа (ОТМД), пакет базы данных в режиме телеобработки КАМА; пакет диалогового удаленного ввода заданий (ДУВЗ).

Аппаратура СТД. Мультиплексоры передачи данных поддерживают взаимодействие ЭВМ с АП через каналы связи с преобразованием и частичным буферированием данных. Они обеспечивают одновременную и независимую работу по многим каналам связи. Алгоритмы работы МПД включают процедуры установления и разъединения связи, идентификации запросов и передачу данных. Порядок выполнения данных процедур инициализируется канальными программами. Взаимодействие МПД и АП состоит в установлении и разъединении связи между ними и обмене информацией через линии связи при использовании АПД.

Если количество передаваемых сообщений в СТД возрастает, то МПД заменяется процессором телеобработки данных, снижающим нагрузку на центральный процессор (ЕС-8371 в ЕС ЭВМ).

АПД включает следующие типы устройств: модемы и устройства преобразования сигналов; вызывные устройства для коммутируемых линий связи, устройства защиты от ошибок (УЗО). АПД делится на низкоскоростную – до 2000 бит/с, среднескоростную – до 4800 бит/с и высокоскоростную – более 4800 бит/с.

Модемы преобразуют двоичные сигналы оконечной аппаратуры (МПД или АП) в модулированные сигналы, передаваемые по каналу связи, и осуществляют обратное преобразование при приеме. Используются частотная и фазовая модуляции.

Для установления связей по коммутируемым линиям применяются телефонные и телеграфные вызывные устройства, подключаемые к модему и оборудованию обработки данных (МПД или АП). Устройства защиты от ошибок организуют передачу данных блоками с циклическим контролем, для которого используются различные образующие полиномы.

Различают три режима передачи информации по каналам связи: симплексный – в одном направлении; полудуплексный – попеременная передача в двух направлениях; дуплексный – одновременная передача в двух направлениях. Различают также синхронную передачу (для передачи блоков данных) и асинхронную (для передачи символов).

Абонентские пункты. АП оборудуются устройствами, позволяющими пользователям в удобной форме вводить задания в удаленную ЭВМ, получать результаты вычислений, а также выполнять действия, связанные с предварительной обработкой данных. В состав АП входят устройства связи оператора с ЭВМ, устройства ввода-вывода информации, память микроЭВМ, а также аппаратура для подключения к каналу связи. В состав АП могут входить также микро- или мини-ЭВМ.

Устройство управления АП реализует алгоритмы его работы и осуществляет подключение и отключение абонентов от ЭВМ и передачу данных между

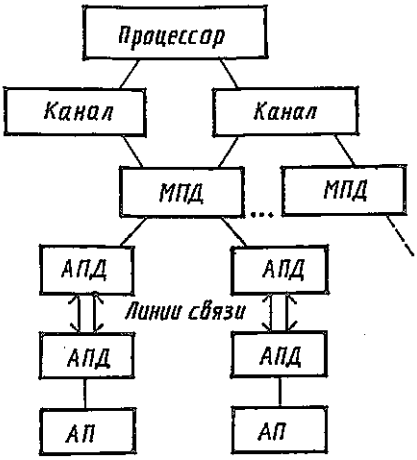


Рис. 4.2.
Структура системы телеобработки данных.

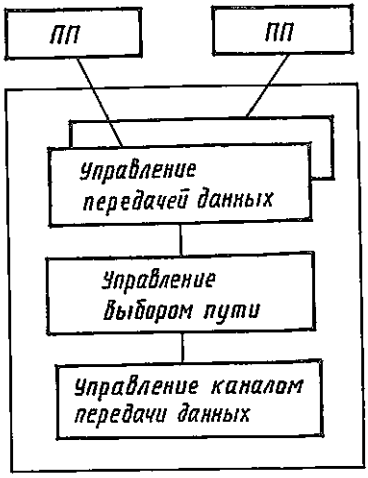


Рис. 4.3.
Основные функции телеобработки, реализованные в ОС.

ними. При приеме информации от ЭВМ выполняется преобразование из кода передачи в код АП, при передаче происходит обратное проебразование. В составе АП можно выделить пульт управления, блок контроля периферийных устройств (ПУ), блок управления подключением ПУ к каналу, блоки управления передачей и соединения с каналом.

Программное обеспечение СТД. Программные средства СТД являются интерфейсом между абонентами и обрабатывающими программами. Они выполняют следующие функции: управление сетью телеобработки данных, управление буферами, редактирование сообщений, управление очередями сообщений, обработку ошибок.

Управление сетью телеобработки данных включает управление каналами связи и АП, сообщениями и буферной памятью. Программы управления каналом связи устанавливают логическое соединение между источником и потребителем данных и определяют направление передачи.

Программы управления сообщениями обеспечивают: формирование символов из бит, блоков из символов, сообщений из блоков, формирование и анализ символов управления (конца блока сообщения, конца передачи), передачу сообщений в соответствии с адресами их назначений, получение сообщений от программ обработки и помещение их в очередь, связь с пультом оператора (формирование сообщений о состоянии системы и ее ошибках), повторную передачу при возникновении ошибок. Выполнение этих функций базируется на объединении ряда АП и каналов связи, идентичных по характеристикам, в группы.

Буферная память в СТД используется для временного хранения данных при передаче их между устройствами, согласования скоростей передачи различных устройств и т. д.

Буферы могут выделяться статически и динамически. Первый метод основывается на задании каналу связи буферного пула.

При динамической буферизации буферный пул (БП) принадлежит всем каналам связи. Для любой операции ввода-вывода буфера из пула выбираются автоматически, заполняются сообщениями, а после их передачи освобождаются и возвращаются в пул.

Приведем пример программы автоматического построения БП.

```
OPEN (CL1,,CL2)  – открытие группы каналов и построение БП
...
CLOSE (CL1,,CL2) – закрытие группы каналов и уничтожение БП
...
CL1 DCB BUFL=50, BUFNO=40  – задание структуры БП
CL2 DCB BUFL=100, BUFNO=10 – задание структуры БП
```

Редактирование сообщений связано с кодопреобразованием и обработкой служебной информации. Программы, реализующие редактирование сообщений, выполняют следующие функции: преобразование кодов (из 5–7-битных кодов АП в 8-битный код ЭВМ и обратно); обработку заголовков сообщений. Заголовок включает разнообразную информацию, касающуюся маршрута, приоритета, видов обработки сообщений и т. д. Между форматом и структурой заголовка и соответствующими программами существует взаимосвязь: структура заголовка определяет порядок расположения макрокоманд.

В СТД очереди сообщений связаны с поступлением сообщений в непредвиденное время и с различной скоростью в основном от АП или при передаче к абоненту из-за отсутствия временной связи с АП. Различают входные очереди из каналов связи, очереди на обработку, обработанных сообщений, выходные очереди сообщений.

В процессе функционирования СТД могут возникнуть ошибки из-за сбоев аппаратуры, программ, действий операторов и т. д. Эти ошибки частично устраняются программами, которые выполняют следующие функции: повторение операций ввода-вывода, при которых произошла ошибка, регистрация неисправляемых ошибок и обработка программных ошибок. На рис. 4.3 показаны основные блоки программ телеобработки.

4.5. ТЕЛЕКОММУНИКАЦИОННЫЕ МЕТОДЫ ДОСТУПА

Базисный метод доступа. Он предназначен для реализации функций обмена между удаленными станциями в ОС ЕС, называется *базисным телекоммуникационным методом* доступа. Метод служит для реализации функций обмена данными между ЭВМ и АП с использованием полудуплексных каналов.

Для программной реализации доступа к удаленным станциям в программе необходимо выполнить следующее: определить технические средства СТД (группы каналов связи, вид связи, буферизацию, код передачи и т. д.); открыть группу каналов связи; отредактировать данные, подлежащие передаче или принятые от АП; использовать МК для передачи сообщений между ЭВМ и АП; использовать МК WAIT для ожидания завершения приема-передачи; проверить завершение ввода-вывода и выполнить при необходимости обработку ошибок; закрыть группу каналов.

Макрокоманды БТМД. Набор макрокоманд БТМД реализует следующие функции: определение системы телеобработки, управление буферами памяти, преобразование кодов, активизацию системы ТОД, прием и передачу сообще-

ний, обнаружение и исправление ошибок, оперативную проверку устройств.

Макрокоманды DCB, DETRMLST служат для определения системы телеобработки. По МК DCB описывается группа каналов связи, а макрокомандой DETRMLST строится абонентский список.

Для создания буферного пула можно использовать макрокоманды BUILD, GETMAIN, GETPOOL. Кроме того, средствами БТМД можно построить буферный пул при открытии группы каналов связи, если в макрокоманде DCB закодированы число буферов и длина. Буферы можно получить из буферного пула и отправить обратно с помощью МК REQBUF – запросить буфер и RELBUF – освободить буфер.

При трансляции сообщений применяется МК ASMTRTAB для построения таблицы трансляции. Таблица используется макрокомандой TRNSLATE непосредственно для преобразования сообщений из кода ЭВМ в код передачи и обратно.

Активизация системы телеобработки производится макрокомандой OPEN, деактивизация – CLOSE . Передача и прием сообщений осуществляются МК READ, WRITE. Синхронизация ввода-вывода выполняется макрокомандами WAIT, TWAIT, которые переводят программу в ожидание события или завершения ввода-вывода.

Для регистрации ошибок служит МК LERB – построение блока ошибок в канале связи и LERPRT – распечатки содержимого счетчика ошибок на консоли оператора.

Для тестового сообщения используется МК ONLST , которая посылает запрос на тест удаленным АП.

Общий телекоммуникационный метод доступа. Он является развитием для СТД метода доступа с очередями ОС ЕС. По сравнению с БТМД это метод более высокого логического уровня. Использование этого метода избавляет программиста от необходимости учитывать особенности алгоритмов передачи данных по каналу связи, программировать опрос и выборку АП, выполнять преобразование кодов, определять конфигурацию сети. Все это выполняется в программе управления сообщениями (ПУС).

ПУС генерируется на основе макрокоманд и модулей ОТМД и работает в разделе с наивысшим приоритетом. ПУС включает следующие секции: активизации и завершения работы системы телеобработки; определения НД, необходимых для работы ПУС; определения конфигурации сети телеобработки, организации обработчиков сообщений.

В *секции активизации* и *завершения работы* определяются основные параметры ПУС. Макрокомандой OPEN открываются наборы данных ПУС. Затем управление передается диспетчеру ПУС, который планирует всю работу, а если ее нет, переходит в состояние ожидания. При завершении работы ПУС управление передается МК CLOSE для закрытия НД, после чего МК RETURN управление возвращается ОС.

Секция определения НД строится с использованием макрокоманд для описания: групп каналов связей, наборов данных, очереди сообщений, системного журнала ПУС и НД контрольной точки.

Секция определения конфигурации сети содержит макрокоманды, описывающие каналы связи, АП и прикладные программы. В ней строятся списки опроса и указываются параметры, определяющие конфигурацию системы.

Секция обработчиков сообщений содержит МК, задающие для каждой группы АП или программ свою обработку сообщений. В секции можно управлять постановкой сообщений в очередь в соответствии с приоритетами, редактировать сообщения, определять маршруты, посылая сообщения в нужную очередь назначения, преобразовывать сообщения из кода ЭВМ в код передачи и обратно.

Макрокоманды ОТМД. Секция активизации и деактивизации ПУС использует макрокоманду INTRO (активизировать ПУС), которая должна быть первой в секции. Макрокоманда OPEN инициализирует НД, используемые в ПУС. Сначала открываются НД очередей сообщений на НМД, затем НД контрольной точки. Для дезактивации открытых НД используется МК CLOSE. Макрокоманда READY кодируется между открытием и закрытием НД и завершает инициализацию ПУС.

Для определения НД служит МК DCB, которая может описывать группы каналов связи, очереди сообщения, системный журнал. Для построения блока связи с прикладной программой применяется МК PCB.

Для управления каналами связи используются макрокоманды: TTABLE – начать абонентскую таблицу; OPTION – описать поля для станции канала связи прикладной программы; TERMINAL – создать элемент абонентской таблицы; TPROCESS – определить прикладную программу; INVLIST – создать список приглашений.

Обработчик сообщений состоит из двух групп макрокоманд: входной и выходной. Выходная группа обрабатывает все сообщения, посылаемые по каналам связи прикладным программам. Входная группа обрабатывает сообщения, поступающие из каналов связи или ПП. Каждая группа состоит из трех подгрупп: заголовков, обрабатывающих заголовки сообщений; буферов, обрабатывающих каждый сегмент сообщения; сообщений, работающих после приема или посылки сообщений.

Программы пользователя в ПУС могут быть открытыми и закрытыми. Открытая может включаться в подгруппы заголовка и буферов обработчика сообщений и использовать МК ОТМД. Закрытая программа вставляется в ПУС как программная секция, в ней нельзя использовать макрокоманды ОТМД.

Прикладные программы выполняются асинхронно с ПУС как отдельные задачи или подзадачи. Для передачи сообщений используются МК GET, PUT или READ, WRITE последовательного метода доступа.

Кроме вышерассмотренных методов, существует виртуальный телекоммуникационный метод доступа. Он, как и ОТМД, освобождает пользователя от написания программы опроса и планирования работы линии связи и включает программные средства, обеспечивающие работу нескольких терминалов, подключенных к одной линии связи. Этот метод преобразовывает входное и выходное сообщения в соответствии с форматом, принятым для передачи в линии, добавляя к данным необходимые заголовки. Для выбора пути к соответствующим прикладным программам вводится соответствующая секция. Управление каналом ввода-вывода, как и в других методах, выполняет планировщик ввода-вывода. Но прикладной программе предоставляется интерфейс более высокого уровня, включающий МК ПЕРЕДАТЬ/ПРИНЯТЬ. Этот метод доступа используется при работе в сети.

4.6. ПРИНЦИПЫ ПРОГРАММИРОВАНИЯ И РАЗРАБОТКИ ПРОГРАММ ДЛЯ СИСТЕМ ТЕЛЕОБРАБОТКИ

Организация программ при работе с одним каналом. При программировании реализуются процедуры управления каналами передачи данных, которые заключаются в следующем: установлении соединения (только для коммутируемых каналов), обеспечении синхронной работы пунктов приема и передачи и определении готовности тракта, в посылке запроса на передачу, передаче данных, завершении соединений.

В СТД центральная станция выполняет обработку и буферизирование сообщений, управление передачей сообщения, обработку ошибок и т. д. Одну часть функций реализует ОС, другую – программы пользователя. Программирование обмена с удаленной станцией при приеме включает следующие процедуры: активизацию канала, установку контакта со станцией и управление приемом данных от нее; перекодировку принятых сообщений из кода передачи в ДКОИ и объединение их в сообщение; удаление символов, обрамляющих сообщение, поиск символа конца передачи, идентификацию сообщения и поиск ПП для его обработки. При передаче осуществляются: прием информации от ПП для передачи в канал; перекодировка сообщения, деление его на блоки и обрамление блоков управляющими символами; передача сообщения удаленному абоненту; управление передачей (повторение блоков при ошибках и т. д.).

Рассмотрим реализацию данных положений при программировании обмена с АП с использованием БТМД. Опишем фрагмент программы определения среды телеобработки, кода передачи, режима работы:

```
TPLINE   DCB DSORG=CX, MACRF=(R,W),DDNAME=LINE1, LERB=LERB4
OTK      DFTRMLST AUTOLST,(4141323205, 0404040404)   списки опроса и выборки, определение DECB
ADR      DFTRMLST OPENLST,(6161323205)
         READ DECB1,T,TPLINE ..., 1, MF=L
         ASMTRTAB RCK7, SCKR     определение таблиц перекодировки
LERB4    LERB 1, (255,50,50,10)  блок регистрации ошибок
IOAREA   DS CL 200   область ввода-вывода
         IECTDECB
         END
```

В этом примере определена среда для работы по синхронному каналу связи с АП ЕС-8504. Макрокоманда DCB указывает использование счетчика регистрации ошибок и макрокоманд READ и WRITE. Блок управления событиями данных DECB1 определяется макрокомандой READ в списковой форме. Заданы два абонентских списка: список опроса OTK и список выборки ADR . Для опроса используется адрес 41, для выборки 61. Обмен будет осуществляться с АП, имеющими адрес 32. Так как для обмена используется код КОИ-7, то заданы две таблицы перекодировки: RCK7 (в ДКОИ) и SCKR (из ДКОИ в русскую часть КОИ-7). Для приема и передачи сообщений определена область IOAREA в 200 байт. Обобщенная схема алгоритма программы обмена для работы по одному каналу связи приведена на рис. 4.4. Программа обмена начи-

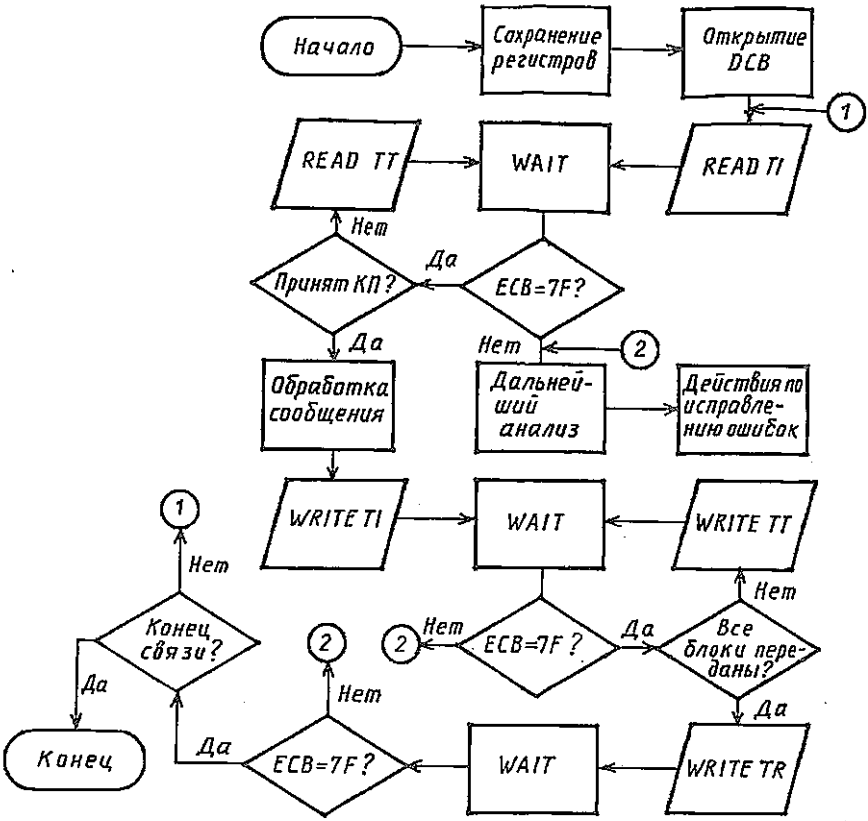


Рис. 4.4.
Схема алгоритма программы обмена по одному каналу связи.

нается с открытия блока DCB и проверки успешности открытия, начальной посылки для установления связи (READ TI), проверки завершения канальной программы (ECB = 7F). При нормальном завершении READ TI (когда установлена связь, считан первый блок) чтение сообщений происходит по макрокоманде READ TT. По принятии конца передачи (КП) начинается обработка сообщения. В случае ненормального завершения канальной программы производится анализ причины и выполняются действия по обработке ошибки (после каждой макрокоманды READ TI и READ TT).

Начало передачи устанавливается по МК WRITE TI (символы связи и первый блок), в случае успешного выполнения осуществляется передача информации по МК WRITE TT. При этом каждый раз происходит проверка канальной программы (анализ причины и обработка ошибок при ненормальном завершении). После передачи всех блоков завершение реализуется макрокомандой WRITE TR с проверкой ее выполнения.

Управление программами и распределение ресурсов. При большом числе абонентов, если не осуществить параллельной обработки сообщений, система

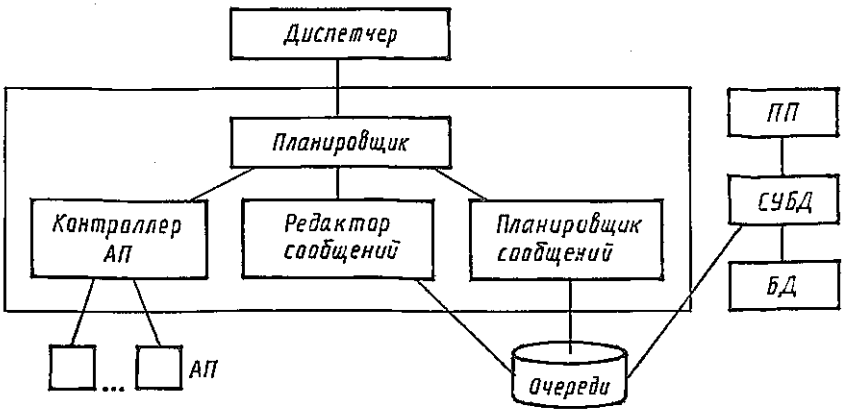


Рис. 4.5.
Упрощенная структура ПО системы телеобработки данных.

через некоторое время окажется перегруженной. Для устранения этого организуют мультиобработку. При этом программа управления сообщениями делается реентерабельной. При нахождении текущей программы пользователя в состоянии ожидания ввода-вывода управление передается следующей программе. Вторая программа выполняется до тех пор, пока не будет задержана операцией ввода-вывода. В этой точке может быть продолжено выполнение первой программы, если операция, вызвавшая ее останов, закончилась. Если операция не закончилась, запускается третья программа и т. д.

При мультипрограммной работе для обмена с каждым АП выделяется своя рабочая область памяти. Каждый АП имеет свою рабочую область, а программа обмена с АП одна. Программа телеобработки должна быть реентерабельной и располагаться в разделе с высшим приоритетом. Фоновые задания получают управление, когда программа телеобработки находится в ожидании ввода-вывода. Реентерабельность программы обмена означает, что для каждого канала связи должны быть свои рабочие области, области ввода-вывода, своя область сохранения регистров, свои списки опроса и выборки (если требуются), а также свой блок управления событиями данных DECB.

Отладка и тестирование программ. Отлаженной считается программа, которая не завершается аварийно и не выдает неверных результатов при любом наборе входных данных. В процессе отладки программа может завершиться нормально или аварийно. Аварийное завершение, как правило, связано с ошибками программиста. В этом случае необходимо убедиться, что при трансляции и редактировании не было ошибок, затем приступить к анализу дампа памяти. При этом определяется команда, вызвавшая ошибку, код завершения, причина аварийного завершения. При нормальном завершении программа может выдать неверный результат в одном случае и верный в другом. Проверка функционирования программы на различных входных НД называется *тестированием*.

Если для тестирования программ телеобработки использовать удаленные АП, то процесс отладки будет медленным. Поэтому целесообразнее для про-

грамм обмена моделировать входную информацию. Если для работы какого-либо модуля входная информация поступает извне, то необходимо разработать управляющую программу, которая обеспечивает передачу модулю входных данных в соответствующей форме.

Структура программного обеспечения. Сеть телеобработки представляет собой комплекс ресурсов различных типов: программы, базы данных, ОС, технические средства. Подсистема передачи данных реализует функции, необходимые для передачи сообщений между процессами, работающими в различных точках сети. К таким функциям относят: управление работой удаленных станций, редактирование входных сообщений для приведения их к стандартам, формирование и обслуживание очередей сообщений, управление буферами, временной контроль системы, управление коммутацией сообщений и т. д.

Упрощенная структура программных средств телеобработки включает диспетчер, управление ПП и подсистему передачи данных (рис. 4.5). Последняя содержит контроллер АП, редактор сообщений, планировщика обработки сообщений и планировщика ввода-вывода.

Контроллер АП управляет обменом сообщений между АП и ЭВМ, устанавливает контакты с АП, получает и принимает сообщения, управляет буферами, исправляет ошибки и т. д. Редактор сообщений выполняет преобразования форматов и кодов, дешифрует заголовки сообщений, редактирует тексты. Планировщик обработки ведет очереди сообщений, выполняет их синтаксический анализ и вызывает обрабатывающие программы. Для реализации протокола обмена могут использоваться таблицы решения. Планировщик ввода-вывода выполняет функции образования подзадач обмена, определения среды телеобработки для абонента, осуществления рестарта в случае аварийного завершения обмена, управления выполнением подзадач и системой с консоли.

Выводы

1. СРВ в ЕС ЭВМ представляет собой программно-языковое расширение базовой ОС ЕС для обеспечения одновременной диалоговой работы пользователей. Организация вычислительных процессов в СРВ условно разделена на пять иерархически связанных уровней: управляющую программу, программу управления разделом РВ, планировщика раздела, монитор и обработчиков команд. Задание основного и вспомогательного кванта времени для задач абонентов выполняет координатор РВ, а планирование вычислительных процессов – диспетчер РВ.

2. Организация вычислительных процессов на уровне пользователя СРВ осуществляется системой команд для создания программ (наборов данных), трансляции, обработки связей, отладки и выполнения. Отдельные команды состоят из подкоманд. Имеются команды пакетной обработки заданий на ЯУЗ ОС ЕС.

3. Организация вычислительных процессов в СТД выполняется на аппаратном и программном уровнях. На аппаратном уровне СТД включает канал ЭВМ, мультиплексор передачи данных (процессор телеобработки), к которому через АПД и линии связи подключается несколько АП. Программное обеспечение СТД выполняет функции управления работой АП, редактирования сообщений, обслуживания очередей сообщений, их синтаксический анализ и исправление ошибок и восстановление ситуации.

4. Базовый метод доступа ОС ЕС в СТД преобразуется в БТМД, а метод доступа с очередями – в ОТМД. Программные компоненты БТМД реализуются набором МК, выполняющих функции активизации и деактивизации СТД, преобразования кодов, организации и ведения буферных пулов, управления каналами, передачи и приема сообщений, исправ-

ления ошибок передачи. На базе МК ОТМД строятся программы управления сообщениями, поддерживающие активизацию и завершение работы СТД, определения ее конфигурации и используемых НД, обработки сообщений.

5. Для работы СТД программист должен описать каналы связи, установку контактов с АП, перекодировку сообщений, их редактирование, вызов ПП, прием информации от ПП для передачи в канал, редактирование и передачу сообщений с контролем, используя БТМД. При работе с ОТМД программист использует либо готовую программу управления сообщениями, либо пишет свою.

Рекомендуемая литература: [8, 24].

Вопросы для контроля

1. Поясните принцип функционирования СРВ.
2. Определите и охарактеризуйте уровни управления в СРВ.
3. Определите виды диспетчеризации в СРВ.
4. Поясните возможности редактирования в СРВ.
5. Дайте порядок трансляции и запуска задач в СРВ.
6. Поясните возможности отладки программ в СРВ.
7. Поясните процесс работы с пакетными заданиями.
8. Охарактеризуйте аппаратное обеспечение СТД.
9. Охарактеризуйте программное обеспечение СТД.
10. Поясните цепочку: прикладная программа–абонент.
11. Сравните БТМД и ОТМД.
12. Поясните построение и использование ПУС.

5. ОРГАНИЗАЦИЯ СИСТЕМЫ ВИРТУАЛЬНЫХ МАШИН

5.1. СОСТАВ И ФУНКЦИОНИРОВАНИЕ СИСТЕМЫ ВИРТУАЛЬНЫХ МАШИН

Общие положения. Система виртуальных машин (СВМ) является операционной системой, предназначенной для управления ЕС ЭВМ "Ряд-2","Ряд-3" и обеспечения одновременной работы пользователей с различными ОС. Под управлением СВМ может функционировать несколько виртуальных машин, имеющих отличную от реальной ЭВМ конфигурацию (рис. 5.1). Работа ВМ происходит под управлением своей ОС. В качестве такой ОС могут использоваться различные версии ОС или ДОС ЕС. Управление ВМ может осуществляться и компонентами монитора виртуальных машин. Реализация в СВМ концепции ВМ обеспечивает принцип параллельной работы пользователей, каждый из которых может работать под управлением собственной ОС. Управление работой ВМ выполняется с пульта, в качестве которого используются терминалы комплекса ЕС-7920, и осуществляется набором диалоговых команд.

В состав СВМ входят: монитор виртуальных машин (МВМ), подсистема диалоговой обработки (ПДО), подсистема дистанционной передачи (ПДП) файлов, подсистема анализа дампов и средства обслуживания и генерации.

МВМ осуществляет распределение и координацию ресурсов реальной ЭВМ и функционирования ВМ. *ПДО* является однопользовательской ОС, предоставляет пользователю возможность создавать, транслировать, отлаживать и выполнять программы на основе системы программирования ОС ЕС. Среда ПДО совместима с ОС ЕС, поэтому программы, разработанные в ОС ЕС, с некоторыми ограничениями можно выполнять в ПДО. Однако ПДО удобнее использовать для создания программ, а ОС ЕС – для их выполнения.

Подсистема дистанционной передачи файлов – специализированная ОС, предназначенная для организации обмена данными между удаленными АП. *Подсистема анализа дампов* представляет совокупность программ, выполняющихся под управлением ПДО и предназначенных для регистрации и анализа ситуаций, приводящих к нарушению работоспособности СВМ. Генерация СВМ может выполняться на выделенной ВМ параллельно с работой пользователей. Ограничения на использование ОС и ДОС ЕС в СВМ связаны с отсутствием средств мультипроцессирования и прямого управления, а также несохранением временных соотношений при выполнении канальных программ и отдельных прерываний.

Языки общения, используемые программистами, определяются видом ОС, функционирующей на ВМ. Для ПДО это язык диалоговых команд, для ОС ЕС – язык управления заданиями или язык команд СРВ. Для системного программиста средствами общения являются языки команд МВМ, ПДО.

Работа виртуальной машины. Для любой ВМ, являющейся функциональным эквивалентом реальной ЭВМ, моделируются все ее физические компо-

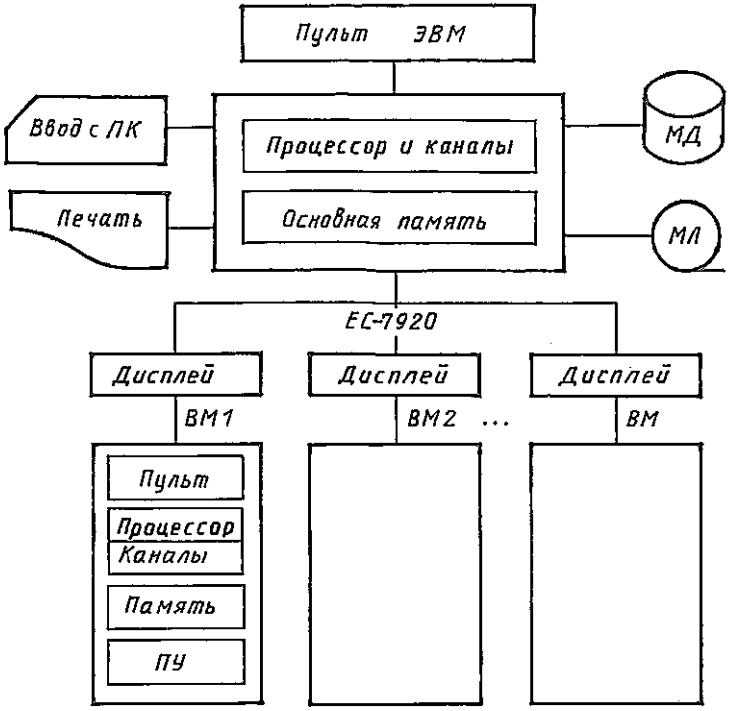


Рис. 5.1.
Моделирование виртуальных машин.

ненты: процессор, память,каналы, устройства управления и периферийные устройства. Принципы работы ВМ (системы команд, прерывания, ввод-вывод, выполнения программ) во многом аналогичны принципам работы ЕС ЭВМ.

Процессор ВМ моделируется путем представления квантов времени процессора реальной ЭВМ. В течение этого времени управление процессором реальной ЭВМ осуществляется текущим ССП виртуальной машины. Прерывания процессора ВМ моделируются программным или микропрограммным способом.

Основная память виртуальной машины моделируется МВМ с помощью аппарата динамического преобразования адресов ЕС ЭВМ и внешней памяти. Практически не накладывается ограничений на емкость основной памяти ВМ (от 8 К байт до 16 М байт). Для сокращения затрат на преобразования адресов памяти ВМ и реальной ЭВМ монитор позволяет закрепить за одной из ВМ часть основной памяти (область V = R).

Периферийные устройства ВМ моделируются монитором следующим образом: закреплением, разделением, накоплением, симуляцией. Суть первого метода заключается в том, что МВМ отдает реальное устройство в полное распоряжение ВМ, причем виртуальный и реальный адреса могут не совпадать. Метод разделения используется только для устройств прямого доступа. С каждым из них связывается участок из нескольких цилиндров, называемый *мини-*

диском, который выступает в роли пакета для ВМ и имеет те же характеристики, но меньший объем.

Метод накопления используется для моделирования устройств единичных записей. Суть метода заключается во введении файла накопления между устройством ВМ и реальным устройством. Записи файла накопления являются образом физических записей (перфокарт, строк печати) реальных устройств.

Методом симуляции МВМ моделирует виртуальный адаптер-канал и псевдотаймер. Виртуальный адаптер применяется для связи между двумя ВМ. Псевдотаймер позволяет ВМ получить дату, время дня, время на работы процессора виртуальной и реальной ЭВМ.

Каналы ввода-вывода включаются в конфигурацию ВМ неявно на основании адресов периферийных устройств. Все периферийные устройства, адреса которых начинаются с одной цифры, считаются подключенными к одному каналу. Канал с адресом О всегда моделируется как байт-мультиплексный. Другие моделируются как селекторные или блок-мультиплексные. Устройства управления включаются в состав ВМ неявно.

Работа ВМ начинается с ее создания монитором, обычно по запросу пользователя. Характер функционирования ВМ определяется состоянием ее процессора (РАБОТА/ОСТАНОВ, СЧЕТ/ОЖИДАНИЕ), а также использованием процессора реальной ЭВМ. Однако имеются и отличия в функционировании ВМ. Нахождение в состоянии СЧЕТ не всегда означает, что процессор реальной ЭВМ выполняет команды ВМ. Это связано с квантованием времени процессора, кроме того, в момент получения кванта процессор может выполнять и команды монитора ВМ для ее обслуживания (моделирование привилегированных команд). Еще одной особенностью работы процессора ВМ является перевод в состояние ОСТАНОВ во время ввода и выполнения команд ВМ, выданных пользователем. В зависимости от использования процессора реальной ЭВМ ВМ может быть в состоянии РАБОТЫ, обслуживания МВМ и ОСТАНОВА. В состоянии работы ВМ находится, когда процессором реальной ЭВМ управляет ССП. В состояние ОСТАНОВА ВМ переводится в случае неготовности использовать процессор реальной ЭВМ либо невыделения кванта времени. В состоянии обслуживания монитора ВМ находится при моделировании устройств, обработки команд МВМ, поступивших от пользователя. Соотношение времен пребывания ВМ в различных состояниях определяется частотой использования привилегированных команд, частотой операций ввода-вывода, количеством одновременно функционирующих ВМ, характером работ, выполняемых на ВМ (диалоговый, пакетный) и т. д.

Управление ВМ. Работа с ВМ производится с пульта терминала ЕС-7927 и начинается с регистрации ВМ. Для регистрации используется команда МВМ LOGON , в которой записывается идентификатор ВМ, затем пароль. Далее пользователю указываются имеющиеся в его распоряжении ресурсы ВМ. Конфигурация ВМ может при необходимости изменяться пользователем в зависимости от емкости основной памяти (DEFINE), удаления каналов и периферийных устройств (DETACH), добавления мини-дисков, абонентских пунктов (DEFINE) и т. д. Предоставляется возможность получения справочной информации о состоянии ВМ (QUERY) и ресурсах реальной ЭВМ (INDICATE). Для установки различных режимов работы ВМ имеется соответствующий набор команд. Завершение работы с ВМ производится по команде LOGOFF.

Общие сведения. ПДО используется для разработки программ, являясь однопрограммной ОС. Среда ПДО совместима со средой ОС ЕС, что позволяет использовать ранее разработанные программы в ПДО, а также выполнять новые ПДО-программы в ОС.

Выполнение работ в ПДО носит диалоговый характер и осуществляется с помощью языка команд. Команды позволяют вызывать системные программы ПДО и программы пользователя. ПДО может работать только на виртуальной машине под управлением МВМ.

Структура ПДО. Подсистема состоит из управляющей и обрабатывающих программ. Управляющая программа работает с ресурсами ВМ (процессором, памятью, периферийными устройствами, файлами) и организует выполнение обрабатывающих программ. Последние осуществляют ряд базовых функций для организации разработки программ и обслуживания ПДО и СВМ в целом. Обрабатывающие программы реализованы в виде обработчиков команд.

ПДО разделена на резидентную (ядро) и транзитную части при размещении в основной памяти. Последняя делится на следующие области: связи (системные таблицы, управляющие блоки, рабочие области), программ ядра, транзитную область, область пользователя и загрузчика. Программы ядра, а также обработчики команд являются реентерабельными, что позволяет организовать разделение одной копии этих программ ПДО для работы с различными ВМ. Первоначально все программы ПДО размещаются на одном системном минидиске. Ядро ПДО всегда занимает фиксированное место, транзитные области располагаются в виде дисковых файлов на оставшейся части системного диска. Кроме того, на системном диске размещаются макробиблиотеки ассемблера, библиотеки объектных модулей и т. д. Системный блок ПДО может разделяться всеми ВМ системы.

Управляющая программа ПДО. Управление процессором осуществляется в однопрограммном режиме без образования подзадач. Порядок смены обрабатывающих программ определяется последовательностью команд ПДО. Обрабатывающие программы теряют управление процессором при возникновении прерываний. Все программы в ПДО выполняются в режиме СУПЕРВИЗОР.

Программа управления основной памятью обеспечивает начальное распределение (рис. 5.2). В процессе работы распределение памяти осуществляется по запросам пользователей. Запросы могут выдаваться явно или неявно. Явные реализуются с помощью предназначенных макрокоманд, неявные – при загрузке или удалении программ из памяти.

При управлении устройствами выполняются следующие функции: управление конфигурацией периферийных устройств, инициирование операций ввода-вывода, обработка прерываний и ошибок при вводе-выводе. Обязательными являются устройство прямого доступа и пульт. Максимально возможный объем включает до десяти НМД, четырех НМЛ, устройства ввода-вывода перфокарт и печатающее устройство. Все устройства, кроме дисков, должны иметь фиксированные адреса (рис. 5.3). Для дисков фиксированными являются адреса системного диска и его расширения. Остальные устройства могут иметь произвольные адреса. Обеспечение описанной конфигурации устройств ВМ заключается в том, что управляющая программа ПДО может выполнять все операции на перечисленных устройствах.

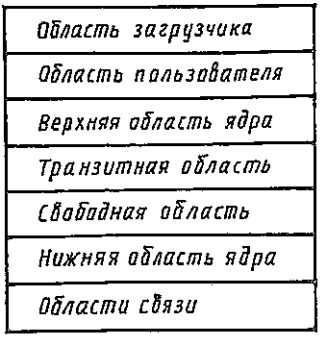


Рис. 5.2.
Схема распределения ОП виртуальных машин для ПДО.

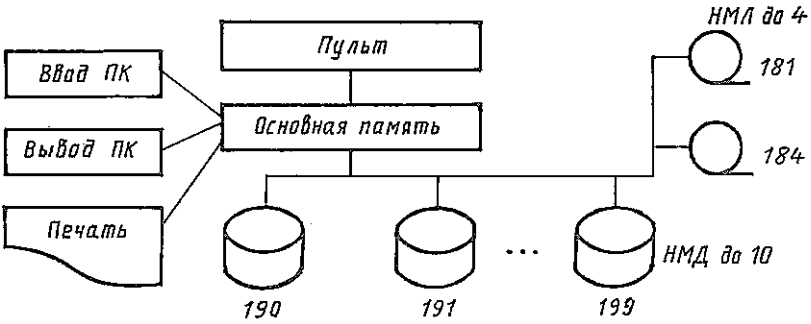


Рис. 5.3.
Конфигурация виртуальной машины ПДО.

Управление файлами. ПДО предоставляет широкий набор средств создания и обработки файлов на внешней памяти. Файловая система ПДО включает логическую и базовую системы. *Логическая файловая система* реализует различные методы доступа к данным и включает макрокоманды ввода-вывода и программы метода доступа. С помощью макрокоманд ввода-вывода выполняются запросы на логические операции обработки данных (чтение и запись). Такие запросы обрабатываются программами методов доступа, обеспечивающими помещение или извлечение логических записей из файлов. Для этого программы методов доступа строят канальные программы и передают их на выполнение супервизору ввода-вывода.

Базовая файловая система выполняет функции распределения памяти на ВЗУ, обеспечивает поиск, защиту и разделение файлов.

С помощью логической файловой системы ПДО обеспечивается обработка файлов с последовательной, прямой и библиотечной организацией и файлов, содержащих логические записи фиксированной, переменной или неопределенной длины. Обработка файлов может осуществляться также моделируемыми в ПДО методами доступа ОС ЕС: BSAM, QSAM, BDAM, BPAM. Файлы на магнитных лентах и устройствах единичных записей могут обрабатываться методами доступа ПДО и последовательным методом доступа ОС ЕС. Обработка одинаковых файлов зависит от форматов, которые бывают в ОС ЕС и ПДО, для чего используются соответствующие методы доступа. Файлы на дисках

ПДО обычно являются последовательными, за исключением системных библиотек.

Обеспечение процедур. Управляющая программа ПДО позволяет автоматизировать ввод группы команд, обеспечивая объединение их в процедуру и последующий вызов с помощью одной команды. Использование процедур дает возможность автоматизировать диалоговую работу и создавать собственные системы команд.

Вызов программы для выполнения в ПДО осуществляется загрузчиком либо программой выборки. Загрузчик вызывается по команде LOAD. При поиске программ сначала просматриваются имена на дисках ПДО, и если программа не найдена, команда передается на обработку МВМ.

Программа связи с пользователем обеспечивает ввод с пульта ВМ команд ПДО и МВМ и вывод сообщений управляющей программы. Наряду с диалоговым режимом в ПДО имеется возможность пакетной обработки. В этом случае задания вводятся с перфокарточного устройства, которое затем выполняется под управлением оператора СВМ.

Команды ПДО. Они могут быть разделены по функциональному назначению на два класса: управляющих и обрабатывающих программ. Команды первого класса управляют функционированием ПДО. К ним относятся: команды установки режимов функционирования ПДО, способов ввода и обработки команд; получения справочной информации о режимах и командах ПДО и МВМ (SET, EXEC, RETURN, HELP и др.); управления доступом к дискам (ACCESS, RELEASE и др.); описания файлов и получение о них информации (FILEDEF, LISTDS, LISTFILE и др.); отладки программ (DEBUG, HO, SO, RO); пуска и прекращения пакетной обработки (PISBATCH, HB).

Команды обрабатывающих программ предназначены для выполнения ряда функций обслуживания. Они делятся на следующие группы: трансляции, редактирования и выполнения (ASSEMBLE, LOAD, INCUDE, START RUN и др.); создания и корректировки файлов (EDIT, UPDATE); создания и обслуживания библиотек (MACLIB, TXTLIB); манипулирования и перемещения файлов (COMPARE, SQRT, ERASE, COPYFILE, DISK, TAPE, TYPE и др.).

Ввод команд определяется состоянием пульта ВМ, действующей средой команд и режимами интерпретации. Когда пульт находится в состоянии СРЕДА МВМ, вводятся только команды МВМ, при состоянии СРЕДА ВМ – команды ПДО либо МВМ. Среда ПДО устанавливается после начальной загрузки, в ней можно выполнять любые команды ПДО и МВМ. При этом могут организовываться среды редактора (EDIT), отладчика (DEBUG), помощи (HELP).

Сеанс работы с ПДО начинается после регистрации ВМ по команде IPL, в которой указывается либо адрес системного диска (190), либо имя хранимой системы ПДО. Затем по команде ACCESS выполняется нестандартная инициализация (активизация дисков 190 и 19E), а по любой другой – нестандартная (активизация дисков 190–192, 19E, поиск и выполнение процедуры PROFILE EXEC). После завершения инициализации ПДО каждый пользователь устанавливает соответствующую среду, лучше используя процедуру PROFILE для автоматизации ее подготовки. Затем он выполняет работы, применяя средства ПДО и МВМ.

Подробно правила использования команд ПДО приведены в [21].

Корректировка исходной программы. ПДО обеспечивает кодирование программы на одном из языков программирования, трансляцию, редактирование, загрузку, отладку и выполнение. При разработке программ возможно использование систем программирования ОС ЕС (если они включены в состав СВМ).

С помощью системы программирования ПДО можно разрабатывать программы, предназначенные для выполнения в ПДО ОС ЕС и для смешанной среды (ПДО и ОС ЕС). Для среды ПДО и смешанной среды используются все возможности ОС ЕС, кроме оговоренных. В среде для ОС ЕС могут применяться все возможности системы программирования, кроме тех, когда разрабатываемая программа включает немоделированные ситуации ОС ЕС. В последнем случае оттранслированный модуль из среды ПДО должен быть перенесен в среду ОС ЕС, там отредактирован и выполнен.

Корректировка может быть выполнена текстовым редактором EDIT или командой UPDATE. Редактор позволяет (как и в СРВ) найти требуемые строки и исправить ошибки. При использовании команды UPDATE необходимо подготовить файлы обновления, в которые включаются управляющие операторы корректировки и корректируемая информация.

Трансляция и загрузка программы. В ПДО трансляция (на примере ассемблера) осуществляется в следующей последовательности: с помощью команды FILE DFF определить файлы, используемые и создаваемые ассемблером; посредством команды GLOBAL определить макробиблиотеки; командой ASSEMBLE протранслировать программу. Последовательность трансляции с других языков аналогична.

В процессе трансляции ассемблер использует ряд файлов: SYSIN – входной, SYSLIB – макробиблиотеки, SYSPRINT – печати, SYSGO – объектный, SYSTERM – диагностический, SYSUT1 – рабочий. Можно применять макробиблиотеки на дисках ПДО и ОС.

Для выполнения оттранслированной программы ее загружают в память ВМ по командам LOAD и INCUDE. При этом объектные модули для загрузки должны находиться в файлах TEXT или TXTLIB (библиотека объектных модулей).

Пуск программы. Он осуществляется после загрузки по команде START. Перед этим файлы, для ввода-вывода которых используются макрокоманды ОС ЕС, должны быть описаны командой FILEDET. При запуске программы по команде START ей можно передать параметры. Если в ПДО осуществляется пуск программы, разработанной в ОС, то необходимо учитывать, что в ПДО и ОС различные форматы списка параметров. Поэтому используется программа-посредник, преобразующая параметры из формата ОС ЕС в ПДО. В процессе передачи управления общие регистры содержат следующую информацию: регистр 15 – адрес основной точки входа, регистр 1 – адрес списка параметров, регистр 13 – адрес области сохранения, регистр 14 – адрес точки возврата.

Отладка программы. Для отладки в ПДО предусмотрены следующие средства: диалоговые отладчики систем программирования ФОРТРАНа, КОБОЛа и ПЛ/1; системные средства для отладки программ на уровне абсолютного и объектного модулей.

К системным средствам отладки относятся средства ПДО (команда DEBUG и ее подкоманды) и средства отладки МВМ. При отладке в ПДО по команде DEBUG предоставляются следующие возможности: установка точек прерывания, при достижении которых происходит останов программы и удаление этих точек (подкоманды BREAK, DELETE); возобновление выполнения программы с указанного адреса и прекращение выполнения (подкоманды G и HX); вывод на пульт ВМ содержимого ячеек памяти, регистров, слова состояния программы PSW, адресного и слова состояния канала CAW, CSW в точках прерывания (подкоманды X, GPR, PSW, CAW, CSW); получение дампа памяти (подкоманды DUMP); изменение содержимого областей памяти, регистров, PSW, CAW, CSW (подкоманды STORE и SET); определение базового адреса для отлаживаемой программы (подкоманда ORIGIN); определение символических имен для использования их при отладке программ (подкоманда DEFINE); протоколирование команд SVC (подкоманда SVCTRACE).

При использовании отладочных средств МВМ можно выполнить следующее: останов (возобновление) программы по указанному адресу (команды BEGIN, ADSTOP); вывод на пульт ВМ содержимого областей памяти, регистров PSW, CAW, CSW (команда DISPLAY); получение дампа памяти (DUMP); изменение содержимого областей памяти, регистров, PSW, CAW, CSW (команда STORE); наблюдение за различными событиями, происходящими при выполнении программы.

Макрокоманды ПДО. Они позволяют выполнять на языке ассемблера ввод-вывод данных для памяти прямого доступа, пульта ВМ, устройств единичных записей, осуществлять редактирование данных, подключать собственные программы для обработки прерываний. С целью организации ввода-вывода используется ряд макрокоманд. Макрокомандой FSREAD реализуется ввод записей в память ВМ из файла ПДО в произвольном порядке с указанием номеров считываемых записей. Для выполнения вывода записей в файл ПДО служит МК FSWRITE, по которой также можно создать новый файл, расширить существующий, добавив записи в его конец, модифицировать файл с изменением записей. Макрокоманды FSOPEN и FSCLOSE используются соответственно для открытия и закрытия файлов ПДО. При открытии файла осуществляется создание блока FSCB (блок управления файлом), если он еще не создан. При закрытии выводного файла обновляется информация оглавления в памяти ВМ. Макрокоманда FSSTATE применяется для проверки наличия файла на указанном диске, а МК FSERASE – для удаления файла.

Макрокоманды ввода-вывода для пульта ВМ позволяют ввести строку (RDTERM), вывести данные на пульт (WRTERM), отредактировать и вывести на пульт или печатающее устройство текст (LINEDIT) и ожидать завершения операций ввода-вывода на пульт (WAITT).

Макрокоманды ввода-вывода для устройства записей позволяют: ввести запись с перфокарточного ввода (RDCARD), вывести запись на перфокарточный вывод (PUNCHC), вывести запись на печатающее устройство без редактирования (PRINT) и с редактированием (LINEDIT). Для работы с НМЛ служат МК считывания и записи данных (RDTAPE, WRTAPE), управления (TAPECTL) создания и обработки стандартных меток (TAPESL).

Для подключения собственных подпрограмм обработки прерывания используются следующие МК: для перехвата внешних прерываний (HNDEXT);

для перехвата прерываний ввода-вывода от указанных устройств (HNDINT); для обработки прерываний по команде SVC с заданными кодами (HND SVC).

Совместимость ПДО и ОС ЕС. Физическое представление программ и данных на носителях в ПДО и ОС ЕС различается. Поэтому для обеспечения возможности использования в одной из этих систем программ и данных другой в ПДО предусмотрены средства перезаписи программ и данных с дисков ОС ЕС на диски ПДО; средства, позволяющие непосредственно применять НД ОС ЕС в ПДО без переписывания на мини-диски; средства перезаписи с дисков ПДО на диски ОС ЕС. Непосредственно в ПДО можно использовать последовательные и библиотечные НД ОС ЕС.

5.4. СТРУКТУРА И РАБОТА МОНИТОРА ВИРТУАЛЬНЫХ МАШИН

Структура МВМ. Основными компонентами МВМ являются управление процессором и памятью, вводом-выводом, восстановлением. Компоненты МВМ состоят из конструктивно завершенных модулей, имеющих имя. В процессе функционирования МВМ использует управляющие блоки, часть из которых создается при генерации, а остальные формируются динамически. Программы и управляющие блоки, полученные при генерации, составляют ядро МВМ, которое размещается в специальной области на резидентном томе. При пуске МВМ ядро считывается в основную память и его резидентная часть сохраняется в течение всего времени функционирования МВМ, модули нерезидентной (страничной) части ядра переписываются во внешнюю страничную память и помещаются в основную только по запросам.

Управление памятью. Основная задача этого компонента — моделирование основной памяти виртуальных машин. Память ВМ является виртуальной и моделируется с помощью средств динамического преобразования адресов (ДПА) ЕС ЭВМ. В соответствии с требованиями ДПА память ВМ делится на сегменты по 64 К байт, каждый из которых включает страницы по 4 К байт. Основная память реальной ЭВМ тоже делится на страницы по 4 К байт. МВМ строит и поддерживает для каждой работающей ВМ таблицы сегментов и страниц, отражающих размещение страниц памяти ВМ в рамках страниц реальной памяти. Эти таблицы используются средством ДПА для определения адресов реальной памяти, соответствующих адресам памяти ВМ.

В основной памяти в каждый момент времени располагается часть страниц памяти ВМ, остальная часть располагается во внешней страничной памяти. Для каждой ВМ строится и поддерживается таблица обмена, отражающая распространение страниц памяти ВМ в страницах внешней памяти. При отсутствии нужной страницы в основной памяти происходит программное прерывание, работа ВМ приостанавливается и выполняется страничный обмен, т. е. редко используемые страницы выводятся во внешнюю страничную память, а на их место вводятся требуемые по запросу страницы памяти ВМ. По завершении обмена возобновляется работа ВМ. МВМ использует также виртуальную память для хранения и замещения страничных модулей МВМ и некоторых рабочих областей монитора. В пределах виртуальной памяти моделируется защита по ключам, аналогичная ОС ЕС.

Реальная память используется для размещения программ и управляющих

блоков МВМ, а также страниц памяти ВМ (см. рис. 5.2). Нулевая страница служит для связи с аппаратурой реальной ЭВМ, содержит постоянно распределенную область памяти, управляющие блоки и некоторые из программ МВМ. Область нестраничных программ включает резидентные модули монитора. Динамическая страничная область предназначена для размещения страниц памяти ВМ в процессе страничного обмена. В свободной области строятся управляющие блоки МВМ. Если эта область окажется заполненной, МВМ выполняет ее расширение за счет динамической области. Наконец, в область трассирования записывается информация о событиях, происходящих в мониторе и реальной ЭВМ, которая может использоваться в случае аварийного завершения МВМ.

Управление процессором. Этот компонент реализует основные функции по распределению времени реального процессора между выполненными задачами и моделированию виртуального процессора.

Задачи, создаваемые МВМ, делятся на системные и задачи ВМ. Системные образуются программами обработки прерываний реальной ЭВМ. Получив управление, такая программа либо сама выполняет обработку, либо создает системную задачу. Задачи МВМ создаются по команде LOGON, причем по одной для каждой машины. Управление задачами базируется на ведение очередей задач к ресурсам ВС. К центральному процессору строятся две очереди системных задач и три очереди ВМ. К другим ресурсам (периферийным устройствам, НД) организуются только очереди системных программ.

Управление очередями по всем ресурсам, кроме процессора, осуществляется соответствующими программами путем представления ресурса очередной задаче и определения очереди для помещения задачи, использовавшей ресурс. Ведение очередей к процессору выполняет планировщик, а обработку очередей – диспетчер.

Планирование задач. Оно обеспечивается для уменьшения страничного обмена, минимизации времени на диалоговые запросы и поддержания уровня мультипрограммирования с балансом вычислений и ввода-вывода. Уменьшение страничного обмена основывается на динамическом отслеживании требований ВМ к основной памяти и обеспечении такого количества активных ВМ, которое не создает большой конкуренции за память. Перевод ВМ из очереди ожидания в активную (RUNLIST) происходит по количественной оценке допустимых и реальных страничных обменов для ВМ. ВМ, удаленные из очереди RUNLIST из-за перехода в состояние останова или ожидания ввода-вывода с пульта ВМ, переводятся в очередь диалоговых ожидающих ВМ E1, а ВМ, удаленные из-за неиспользования ими интервала планировщика, помещаются в очередь ожидающих недиалоговых ВМ E2. Состояние ВМ при переводе их из очередей E1, E2 в активную очередь RUNLIST сохраняется. Диалоговость отражается на длительности интервала, который предоставляется ВМ в активной очереди. Этот интервал для диалоговых ВМ равен 300 мс, для недиалоговых – 2 с (для ЕС 1035, ЕС 1036, ЕС 1045). Для ЕС 1061, ЕС 1065, ЕС 1066 это время соответственно 100 мс и 1 с.

Приоритетность обслуживания диалоговых ВМ обеспечивается первоочередным просмотром очереди E1 при пополнении активной очереди. Пользователь может планировать некоторые параметры задач ВМ, применяя команду INDICATE.

Диспетчеризация задач. Она состоит в выборе задачи и передаче ей управ-

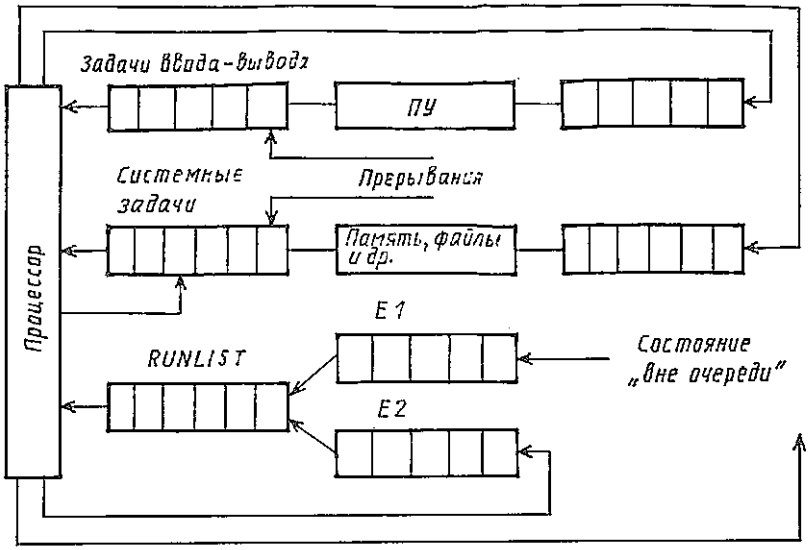


Рис. 5.4.
Схема диспетчеризации и планирования МВМ.

ления ЦП. Диспетчер последовательно просматривает три очереди: системных задач ввода-вывода, других системных задач и очередь RUNLIST (рис. 5.4). Системная задача, получив управление, будет использовать ЦП до тех пор, пока не будет завершена или переведена в очередь к другому ресурсу. Переход к следующей очереди происходит, если все предыдущие пусты.

ВМ из очереди RUNLIST применяют процессор в режиме квантования. Каждая задача получает квант времени (50 мс), который используется полностью или частично (из-за прерывания). В первом случае при повторной диспетчеризации задаче выделяется больший квант (200 мс), в случае прерывания повторный – квант 50 мс. При реализации полного кванта ВМ перемещается к концу очереди, при неполной ВМ не меняют своего положения в очереди.

Моделирование процессора ВМ. Оно происходит представлением времени ЦП задаче ВМ. Для вмешательства монитора используется модификация PSW ВМ. При любом прерывании задачи ВМ управление получает монитор, который в зависимости от причины либо обрабатывает его путем моделирования, либо переводит ВМ в состояние ожидания. Монитор обеспечивает моделирование практически всех средств реального процессора: машинных команд, системы прерываний, регистров, динамического преобразования адресов, таймера, регистрации программных событий.

Управление устройствами. Для управления устройствами МВМ выполняет функции распределения реальных устройств, моделирования устройств ВМ, планирования и запуска операции ввода-вывода, обработку ошибок.

Тома в СВМ делятся на тома монитора и ОС для ВМ. Тома МВМ включают резидентный, собственный и мини-диски. Резидентный содержит ядро, с которого происходит загрузка монитора, а также области теплого пуска, контроль-

Рис. 5.5.
Размещение СВМ ЕС на двух томах ЕС-5066.

ной точки и регистрации ошибок. В области теплого пуска и контрольной точки помещается информация, необходимая для восстановления работы после нормального или аварийного завершения СВМ с заданного адреса. В область регистрации помещается информация об ошибках процессора, канала ввода-вывода и периферийных устройств.

Собственный том МВМ содержит области оглавления СВМ ЕС, страничного обмена, хранимых систем и сегментов, временных мини-дисков. Собственный том может использоваться также в качестве резидентного. На рис. 5.5 приведен пример размещения МВМ.

Реальные периферийные устройства распределяются в СВМ ЕС между монитором и ВМ двумя способами: автоматически и вручную. *Автоматическое* распределение происходит при инициализации МВМ и регистрации ВМ, *ручное* осуществляется оператором. Оператор выполняет закрепление за МВМ или ВМ требуемых периферийных устройств, а также может освобождать закрепленные за МВМ устройства: диски — в случае равенства нулю счетчика закреплений и НМЛ — всегда. Освобождение устройств, закрепленных за ВМ, происходит в любой момент времени.

После получения запроса МВМ определяет свободный путь к данному устройству. Если путь найден (через различные каналы), запрос инициируется,

иначе он ставится в очередь в соответствии с порядком его поступления. После освобождения некоторого пути МВМ инициирует запрос из соответствующей очереди.

В результате машинных ошибок и ошибок ввода-вывода управление получают программы средств восстановления, которые в зависимости от типа ошибки позволяют продолжить работу, завершить функционирование одной из ВМ или аварийно всей системы.

5.5. ОБСЛУЖИВАНИЕ И ГЕНЕРАЦИЯ СВМ

Операторное обслуживание. Обеспечивая работу системы, оператор выполняет функции по запуску, останову СВМ, управлению ресурсами реальной ЭВМ, обслуживанию пользователей ВМ, управлению накоплением. Функции управления осуществляются с пульта управления выделенной оператору ВМ. Ему предоставляются в распоряжение главный пульт и несколько сменных. С помощью команды DISCONN и LOGON оператор может переключить свою ВМ на любой пульт, доступный монитору. Протокол работы на пульте оператора записывается в специальный файл, который может быть распечатан по окончании работы или в случае его закрытия.

Пуск СВМ включает подготовку необходимых устройств (установку требуемых томов), начальную загрузку СВМ, ее специализацию (таймера, пульта оператора, памяти, периферийных устройств), восстановление состояния после предыдущего сеанса. Различают холодный пуск (начальный), теплый (восстановление после предыдущей работы), пуск с контрольной точки.

Завершение работы СВМ может быть нормальным или аварийным. *Нормальное завершение* происходит по команде SHUTDOWN, при этом оканчиваются сеансы всех ВМ, закрываются файлы, останавливается монитор, создаются данные для теплого пуска. *Аварийное завершение* может наступить в результате отказа аппаратуры, ошибок МВМ, неправильных действий оператора. При этом сохраняются данные для теплого пуска.

Важной функцией оператора является управление ресурсами реальной ЭВМ: процессора, основной памяти, периферийных устройств, а также координация использования этих ресурсов пользователями. Для этого имеется набор команд управления АП, работы с МВМ.

Обслуживание пользователей ВМ включает обеспечение их ресурсами, организацию взаимодействия, представление сервиса для работы. Для посылки сообщения используется команда MESSAGE. Принудительное завершение работы ВМ осуществляется по команде FORCE.

В число функций, выполняемых оператором по накоплению (вывод файлов пользователей на реальные устройства и создание входных файлов), входят: пуск и останов работы устройств единичных записей; управление порядком вывода файлов; управление реальными устройствами единичных записей.

Виды и этапы генерации СВМ. Генерация представляет собой создание конкретного варианта СВМ. Различают полную и частичную генерации. Полная включает создание оглавления СВМ, генерацию ядра МВМ, ПДО и подсистемы дистанционной передачи файлов. Частичная генерация содержит отдельные виды работ. Первая полная генерация выполняется на базе дистрибутивной системы. Последующие полные и частичные генерации могут выполняться на базе

варианта СВМ. Генерация системы включает этапы планирования и выполнения генерации.

На этапе планирования конкретного варианта СВМ анализируются условия эксплуатации, круг пользователей, конфигурация ЭВМ. В результате анализа определяются возможности СВМ, режимы и параметры функционирования, ВМ пользователей, и также средства и последовательность выполнения генерации.

После планирования полной генерации подготавливаются файлы описания СВМ (оглавления СВМ, описания МВМ, управляющий, описания хранимых систем и описания ПДО), схема размещения генерируемого варианта на томах, план выполнения генерации на ЭВМ.

Дистрибутивные системы. Это совокупность программных материалов для поставки пользователю, включающих стартер и мини-диски. Стартер представляет конкретный вариант СВМ, подготовленный специально для генерации. В оглавлении стартера описана единственная ВМ с идентификатором SUSPRG, используемая при генерации. Мини-диски содержат компоненты СВМ в виде, пригодном для генерации конкретных вариантов, а также программные средства генерации СВМ. Мини-диски включены в конфигурацию ВМ SUSPRG с адресами 190 и 194. На мини-диске 190 находятся компоненты ПДО и анализа дампов, а также средства генерации всех компонентов СВМ. На 194-м мини-диске размещаются компоненты МВМ и ПДП, а также их средства генерации. Дистрибутивная система поставляется на дистрибутивных (двух) магнитных лентах. На них расположены три файла. Первые два файла содержат автономные программы обслуживания, обеспечивающие подготовку томов и восстановление дистрибутивной системы с ленты на диски. Третий файл включает образ полного тома диска (ЕС-5061) или его части (ЕС-5066).

Планирование генерации. Функциональные возможности генерируемого варианта СВМ определяются включаемыми компонентами. Обязательным является МВМ, настроенный на конфигурацию ЭВМ, режимы и параметры работы. ПДО не является обязательной, поскольку работы могут выполняться другими ОС. Необязательными являются также ПДП и анализ дампов.

При описании конфигурации ЭВМ задается объем основной памяти (файл DMKSYS), перечисляются все каналы, устройства управления, периферийные устройства, средства телеобработки в файле DMKRIO.

При планировании перечисляются идентификации: ВМ главного оператора, ВМ для обработки дампов, ВМ для обработки учетных записей, времени. Указываются режимы монитора для сбора информации о функционировании СВМ, режимы контроля доступа, описывается ядро МВМ и виртуальные машины пользователей, а также размещение СВМ на томах прямого доступа. Наконец, производится оценка основной и внешней памяти на томах.

Процесс генерации. Он включает следующие основные шаги: подготовку томов, восстановление на них дистрибутивной системы с МЛ, загрузку стартера СВМ, создание оглавления СВМ, генерацию ядра МВМ, генерацию ПДО, создание хранимых сегментов ПДО. После завершения генерации выполняется проверка функционирования варианта СВМ. Полная генерация занимает примерно два часа времени без подготовки томов. При генерации СВМ использу-

ются команды МВМ, команды ПДО, средства генерации и программы обслуживания.

Выводы

1. ОС СВМ ЕС представляет дальнейшее развитие базовой ОС ЕС для обеспечения одновременной диалоговой работы нескольких пользователей под управлением различных ОС на одной ЭВМ. Организация вычислительных процессов в СВМ поддерживается средствами монитора ВМ, подсистем диалоговой обработки и передачи файлов.

2. ПДО является однопользовательской ОС, функционирующей на ВМ. Она включает управляющую программу для работы с ресурсами ВМ (процессора, памяти, периферийных устройств и файлов), управления вычислительным процессом на ВМ, а также обрабатывающие программы.

3. Обрабатывающие программы ПДО вызываются командами пользователя. Вычислительные процессы, запускаемые командами, позволяют создавать тексты программ и данных, транслировать и загружать программы в ОП, отлаживать и выполнять их. Пользователь может работать с макрокомандами для организации файлов, ввода-вывода на пульт ВМ, обработки прерываний. Совместимость ПДО и ЕС обеспечивается средствами перезаписи наборов данных на дисках.

4. Монитор ВМ организует вычислительные процессы для выполнения функций работы с виртуальной памятью с целью выделения ее части каждой ВМ, планирования работы ВМ путем построения очередей активных, диалоговых ожидающих и недиалоговых ожидающих задач; диспетчеризации задач путем обслуживания в режиме РВ трех очередей: системных задач ввода-вывода, других системных задач и активных программ ВМ; работ с периферийными устройствами.

5. Оператор для организации вычислительного процесса в СВМ использует набор команд для запуска системы, управления ресурсами, обслуживания пользователей ВМ, завершения работы. Процесс генерации (полной или частичной) может выполняться на одной ВМ с использованием команд монитора ВМ, ПДО и средств генерации.

Рекомендуемая литература: [21].

Вопросы для контроля

1. В чем заключается моделирование процессора и памяти в СВМ?
2. В чем заключается моделирование периферийных устройств в СВМ?
3. Перечислите состав виртуальной памяти ПДО.
4. Объясните управление процессором в МВМ.
5. Объясните управление памятью и устройствами в МВМ.
6. Поясните сущность планирования в мониторе ВМ.
7. В чем заключается диспетчеризация в МВМ?
8. Какие функции и средства использует оператор СВМ?
9. Поясните процесс генерации СВМ.
10. Поясните состав и работу управляющей программы ПДО.

6. ОРГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ В ДИАЛОГОВЫХ СИСТЕМАХ

6.1. СОСТАВ И ВИДЫ ДИАЛОГОВЫХ СИСТЕМ

Виды диалога. Диалоговыми системами (ДС) являются языково-программные средства, обеспечивающие работу пользователя по решению задач на ЭВМ в режиме оперативного обмена: команда пользователя – выполнение действий ЭВМ с выдачей соответствующих сообщений. ДС, как правило, на аппаратном уровне поддерживаются видеотерминалами. Решение задачи с использованием ДС может включать следующие этапы: уточнение условий (формулировка) задачи; выбор (конструирование) алгоритма; ввод данных и решение задачи (слежение за ее прохождением); вывод и просмотр результатов, анализ качества решения и конечный контроль. При этом каждый из этапов может быть разделен на более мелкие.

Под *диалогом* понимают обмен информацией между пользователем и ЭВМ, преследующий вполне определенные цели. Структуру диалога принято представлять в виде дерева, называемого *разговором*. При этом в узлах находятся так называемые обмены, а дуги соответствуют логическим связям между обменами. Под *обменом* понимается вывод информации системой и возможный ответ пользователя. В зависимости от видов обмена различают следующие основные виды диалога: меню, командный, заполнение экранных форм, смешанный и т. д. Под *меню* понимается выдаваемый пользователю перечень ответов, действий, из которых он выбирает одно. Частным случаем меню является диалог типа да/нет. *Командный диалог* включает наличие процедур, каждая из которых вызывается соответствующей командой пользователя. *Заполнение экранных форм* заключается в реакции пользователя на представляемую ему информацию в виде таблицы, которую необходимо заполнить, возможно, по предлагаемому образцу. *Смешанный диалог* включает комбинации вышерассмотренных форм.

В настоящее время интенсивные работы проводятся над построением диалога, понимающего фразы на естественном языке. Этот вид диалоговых взаимодействий, называемый *интеллектуальным*, требует морфологической, синтаксической и семантической обработки текстов, выполняемой так называемым лингвистическим процессором, средств представления элементов знаний о внешнем мире как языковых, так и неязыковых (модель мира), и моделей формального описания языка, с которыми работает ЭВМ [16].

Типы ДС. По своему назначению ДС делят на проблемно-ориентированные и общего назначения. *Проблемно-ориентированные системы* предназначены для узкого круга лиц, решающих задачи в конкретной предметной области. Как правило, такие системы имеют фиксированные формы диалоговых взаимодействий, трудно перестраиваются на новые приложения, требуют определенной подготовки для их освоения и сопровождения. *ДС общего назначения* допускают настройку на решение задач предметной области.

По используемым языковым средствам ДС делят на системы, включающие командные языки, логические, реляционные и ролевые. Первые содержат набор команд, которые позволяют пользователю выполнять создание, запуск решение и анализ задачи, а также управлять системой. ДС, использующие логические языки, включают для поддержки формальную систему логического типа (исчисление высказываний или исчисление предикатов первого порядка, многозначные логики или модальные исчисления). Синтаксис языка задается правилом построения правильных синтаксических выражений.

Для реляционных языков характерным является введение конечного множества бинарных отношений R , с помощью которых передаются смысловые связи между элементами языка. Обычно рассматриваются языки, в которых заданы конечные множества элементов языка и отношений R . Ролевые языки опираются на специальные конструкции, называемые *ролевыми фреймами*, и используются для представления знаний в интеллектуальных ДС. Однако в таких системах достаточно сложное ведение эффективных процедур преобразований, а теория подобных языков находится в стадии развития. Рассмотрим более подробно ДС общего назначения с командными языками. К таким системам относятся ДС, имеющие похожие принципы построения: ДЖЕК, ОКО, ФОКУС, ПРИМУС. Рассмотрим одну из них.

ДС коллективного пользования. В последнее время на ЕС ЭВМ широкое распространение получила диалоговая система ПРИМУС [7], разработанная в Московском инженерно-физическом институте. Система использует дисплейные станции различного типа (ЕС-7920, ЕС-7906 и др.). ДС ПРИМУС позволяет работать нескольким пользователям, предоставляя им большой набор функциональных команд для создания и корректировки программ и текстов , запуска задач, просмотра результатов и управления системой. Каждый пользователь имеет в распоряжении рабочий (РНД) и личный (ЛНД) наборы данных. В рабочем наборе информация сохраняется на время сеанса работы, в личном – неограниченно долго.

В рамках собственного программного процесса пользователь может инициировать выполнение как диалоговой, так и любой другой программы. При этом на ее структуру не накладывается никаких ограничений. Для диалоговой программы обеспечение взаимодействия с терминалом базируется на специальных средствах ввода-вывода. В действительности программа взаимодействует не с физическим, а с виртуальным терминалом (ВТ), имеющим свой логический номер. Функции отображения физического терминала на виртуальный и обратно реализуются программой управления сообщениями (ПУС). Многообразие используемых терминалов как локальных, так и удаленных обеспечивается включением нескольких ПУС в состав ядра системы. По аналогии с терминалами ДС обслуживает несколько печатающих устройств для получения твердой копии, при этом они поддерживаются специальными программами управления выводом (ПУВ). Набор последних может пополняться для расширения класса обслуживания физических устройств.

Потенциально каждый пользователь может получить доступ к любому НД системы. Подобная свобода не всегда является допустимой, поэтому в состав системы включена подсистема защиты от несанкционированного доступа.

Структура ДС ПРИМУС. Система функционирует в рамках одношагового задания, выполняющегося под управлением ОС ЕС, позволяя поддерживать

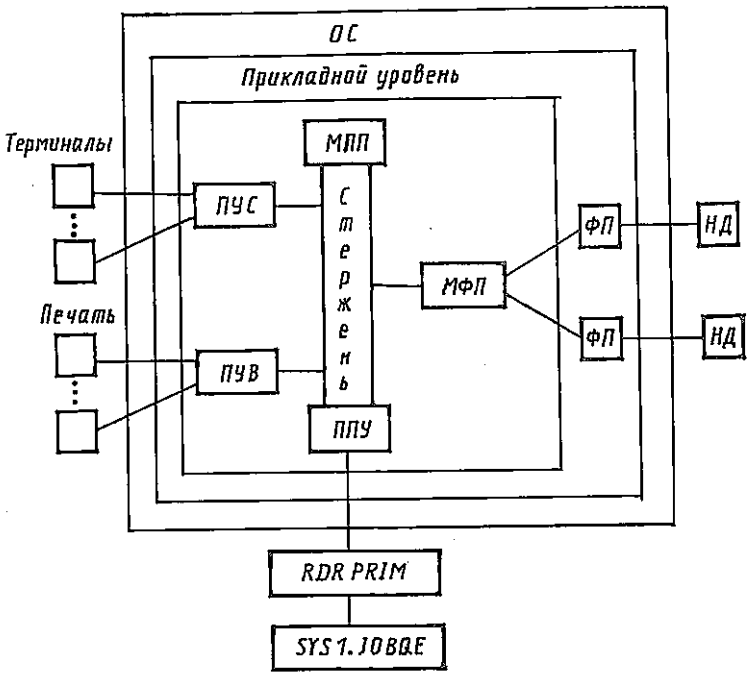


Рис. 6.1.
Структура ПО диалоговой системы ПРИМУС.

связь с другими заданиями ОС. Программное обеспечение системы представляет собой совокупность параллельных программных процессов, выполняющихся в рамках подзадач ОС. Общую структуру системы делят на ядро и прикладной уровень (рис. 6.1). Ядро инициализируется в момент начальной загрузки системы и занимает постоянный объем памяти в течение всего времени работы. Ядро выполняет следующие функции: управление программными процессами системы, программами пользователей, терминалами, печатающими устройствами, заданиями, передаваемыми на выполнение в пакетном режиме. В связи с этим в состав ядра входят следующие компоненты: монитор программных процессов (МПП), монитор функциональных программ (МФП), планировщик первого уровня (ППУ), а также ПУС и ПУВ.

Для объединения параллельных процессов ядра в единую систему используется дополнительный программный процесс, называемый *стержнем*. Последний не имеет непосредственной связи с внешней средой, является пассивным и обеспечивает обмен сообщениями между программными процессами. Конструктивно стержень состоит из набора структур данных, программных модулей для обработки этих структур и диспетчера. Стержень активизируется каждый раз, когда к нему обращается программный процесс для передачи некоторого внутреннего сообщения с указанием пункта его назначения. Результатом работы стержня является изменение его структур данных и, возможно,

регистрации события, приводящего к активизации другого программного процесса. Одновременно поступающие сообщения обслуживаются диспетчером.

Главным программным процессом ядра является МПП, который выполняется в рамках головной задачи шага заданий и несет ответственность за порождение других программных процессов, обработку аварийного завершения и восстановление их работы. Все программные процессы ядра порождаются в момент инициализации системы, процессы пользователей создаются в начале сеанса и уничтожаются в момент его окончания. В рамках программных процессов пользователей работает МФП, который, являясь реентерабельным, используется во всех программных процессах. В момент создания программного процесса пользователя ему дается ВТ, с которым может взаимодействовать МФП.

В начале работы МФП определяет имя и права пользователя, в случае успешного завершения монитор переходит к интерпретации команд системы. Если команда выдана верно, МФП загружает требуемую функциональную программу (ФП) в основную память и передает ей управление. Эта ФП вступает в диалоговое взаимодействие с пользователем, она может вызвать личную диалоговую программу или произвольную и передать ей управление. Завершение ФП обрабатывается средствами МФП. Тогда освобождаются все ресурсы, полученные ФП, и закрываются все НД, используемые в ФП.

Заявки, поступающие от ФП на выполнение заданий в пакетном режиме, обрабатываются ППУ, который является самостоятельным программным процессом ядра. ППУ поддерживает внутреннюю очередь заданий, которая обслуживается по принципу FIFO. Задания из этой очереди передаются планировщику ОС путем запуска специальной процедуры системного ввода RDRPRIM, но не более одной одновременно.

Система команд ДС ПРИМУС. Для входа в систему пользователь указывает свое имя, согласованное с администратором, а при необходимости и пароль. После успешного вхождения в систему вся дальнейшая работа осуществляется через систему команд. Последняя включает внешние и внутренние команды (подкоманды). С помощью внешних команд пользователь может выполнить любую ФП системы, набрав ее код и, если необходимо, операнды. Отдельные ФП, вступая в диалог с пользователем, применяют подкоманды. После окончания выполнения ФП на экране появляется сообщение: ПОСЫЛАЙТЕ КОМАНДЫ. Для описания и ознакомления пользователя с системой команд можно воспользоваться командой HELP. В результате на экран выводится список стандартных функциональных команд и их краткое описание.

При работе с РНД пользователь имеет возможность вводить информацию с экрана (INPT), корректировать ее (CORR), сохранять ее в ЛНД или копировать из него в рабочий НД (SAVE и COPY). При работе с ЛНД пользователь имеет возможность создавать личный НД (DSET), просматривать и корректировать его (LOOK), удалять (SCRT), переименовывать его (RENM), осуществлять обмен между наборами (MOVE), просматривать оглавление личной библиотеки (CONT).

При работе с программами пользователь может вызвать личную диалоговую программу LINK, выполнить одношаговое задание в режиме СРВ RUN, определить DD-предложения. Для запуска заданий в пакетном режиме используется команда EXEC. Просмотр результатов осуществляется по команде

SOUT. Слежение за состоянием ОС выполняется командами: DASD – просмотр списка доступных томов прямого доступа; VTOC – просмотр оглавления тома; CONS – просмотр содержимого буфера главной дисплей-консоли; OPER – вызов монитора команд оператора; D A, D Q – получение информации об активности и состоянии очередей и др.

Для программиста ПРИМУС является удобным инструментом при вводе, редактировании и отладке программ. Однако для пользователя-непрограммиста система не содержит достаточно развитых средств, таких, как генерация пусковых пакетов на ЯУЗ, обеспечение справочной информацией по прикладным программам, ввод данных для программ и т. д. Вместе с тем открытость системы (список команд может быть расширен, если реализованы соответствующие функциональные программы) и наличие подпрограмм связи с ВТ позволяют рассматривать ее как базовое программное обеспечение специализированных ДС (управление, проектирование), а также ДС для пользователей-непрограммистов.

Для реализации доступа к терминалу система включает три основные программы, вызываемые из программ, написанных на языках ассемблер и ПЛ/1: PLENV – для получения числа строк экрана и логического номера ВТ; PLGET – для получения сообщения с терминала; PLPUT – для вывода сообщений на терминал. Более подробно с системой ПРИМУС можно познакомиться в [7]. В дальнейшем будут рассмотрены подходы к построению ДС на базе системы ПРИМУС, хотя многие принципы могут использоваться при создании ДС на других ЭВМ, как с базовым монитором, так и без него.

6.2. ДИАЛОГОВЫЙ ЗАПУСК ЗАДАЧ И АНАЛИЗ РЕЗУЛЬТАТОВ

Построение сценариев диалога. Основными требованиями при решении задач в диалоге на основе имеющегося пакета программ являются: получение справочных данных по предметной области с возможностью легко корректировать эти данные; ввод и корректировка исходных данных с подсказками системы; запуск и слежение за выполнением отдельных компонентов задачи: анализ результатов с возможностью полного или частичного сохранения. С этой целью минимальная конфигурация проблемной ДС, построенной на базе диалогового монитора (ПРИМУСа или другого), должна включать следующие подсистемы: справочную, решения, слежения и анализа, которые можно настраивать на конкретные прикладные программы предметной области путем изменения сценариев ДС.

Под *сценарием* будем понимать связанный набор обменов, определяющий действия системы в любой возможной ситуации диалогового процесса. Таким действием может быть получение справочной информации, ввод данных, просмотр кадра результатов и т. д.

Сценарий разрабатывается проектировщиком диалога, возможно, на естественном языке и помещается в базу сценариев или в текст программ. В последнем случае говорят о "жестком" диалоге, модификация которого требует изменения текста программ. Рассмотрим пример сценария при работе с командой LOOK системы ПРИМУС, для простоты считая, что она единственная в системе. Имеем алгоритм: запросить команду, если не LOOK – ошибка; если

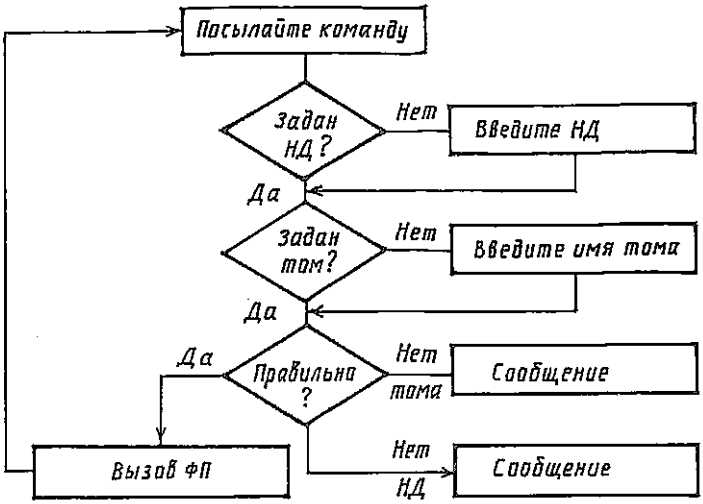


Рис. 6.2.
Граф диалога при выполнении команды LOOK.

НД не указан, запросить имя НД; если том не определен, запросить имя тома; если нет такого тома или на данном томе нет указанного НД – ошибка, перейти к началу; показать запрошенный НД и перейти к началу. Этот сценарий наглядно изображается графом (рис. 6.2). В ходе рассмотренного диалога осуществлялись обмены для выдачи так называемых предметных переменных (имя НД и имя тома), а также шаги, состоящие в выполнении отдельных программ с учетом значений полученных переменных.

Граф сценария – удобное средство представления сценариев широкого круга ДС. Вершиной начала является главное меню, определяющее возможности данного сценария. Реализация сценария заключается в выполнении предписаний, содержащихся в вершинах графа (получение предметных переменных, выполнение программ, условный или безусловный переход к следующей вершине). Моделью такого диалога является конечный автомат, состояния которого – вершины графа, входные переменные – ответы пользователя, а выходные переменные – выполнение предписания, указанного в вершине.

Определим следующие типы вершин: Меню, Сообщение, Таблица, Программа, Процедура, Анализ. Под *Меню* будем понимать вершину с одной системной переменной – Ответ, реализующий выбор одной из возможных альтернатив. *Сообщение* – вершина, не требующая действия. *Таблица* – вершина, присваивающая значения группе предметных переменных. *Программа* – вершина, реализующая выполнение прикладной программы, имеет доступ к значениям предметных переменных. *Процедура* – вершина, выполняющая арифметические и логические операции, анализ значений предметных переменных, определение следующей вершины, а также выполнение функций базового монитора (например, команд системы ПРИМУС). *Анализ* – вершина, осуществляющая просмотр наборов данных, предназначенных для вывода на печать, в

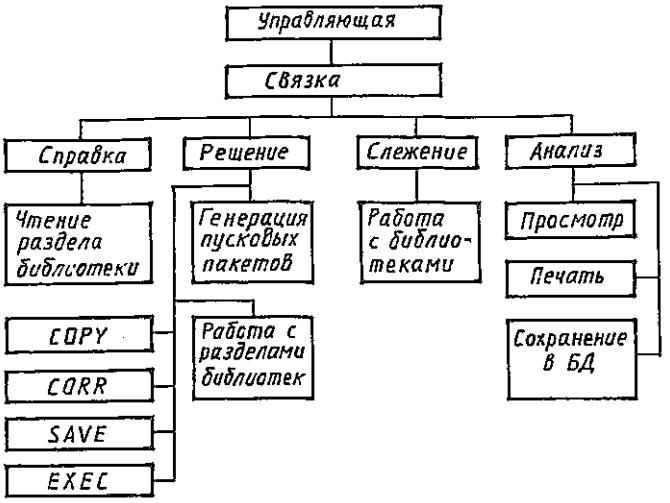


Рис. 6.3.
Структура системы диалогового решения задач.

БД, полученных в результате работы проблемных программ. Описание вершин графа сценария состоит из секций, при этом набор секций зависит от типа вершины.

Типы секций графа диалога. Существуют следующие виды секций: предметных переменных, процедурная, экрана, характеристик программы и DD-предложений. В *секции предметных переменных* каждой переменной присваивается имя, определяются ее характеристики (тип, длина, область действия: локальная – для одной вершины и глобальная – для всего диалогового процесса; смысл на естественном языке, область применения). Такая секция присутствует в вершинах типа Таблица (для ввода предметных переменных) и в вершинах типа Программа (для описания параметров).

Процедурная секция может входить во все вершины, а также представлять самостоятельную вершину типа Процедура. *Секция экрана* содержит постоянную информацию, которую необходимо показать на экране и которая входит в описание любой диалоговой вершины. Вершина типа Сообщение содержит только данную секцию.

Для вершины типа Программа дополнительно вводятся следующие секции. *Секция характеристик* программ содержит имя, координаты программы, максимальное время выполнения, способ вызова (непосредственный, как подзадача, выполнение в рамках отдельного задания), способ передачи параметров (по стандартным соглашениям ОС или через поле PARM языка управления заданиями и т. д.). В *секции DD-предложений* описываются операторы DD, необходимые для выполнения программы.

Структура ДС решения задач. Используя определенные выше понятия, рассмотрим организацию диалоговых процессов при разработке ДС решения и анализа задач (рис. 6.3). Справочная система обеспечивает пользователя иерархически расположенной информацией для проблемной области, т. е. о назначе-

нии, особенностях и возможностях конкретного пакета программ. Вся справочная информация разбивается на разделы (1–3 страницы), каждая по 19 информационных строк (максимум). Остальные строки экрана дисплея являются служебными для сообщений действий пользователя и задания адресов перехода к следующей вершине по дереву. Дерево справочной информации строится из вершин двух типов: Меню и Сообщения. В корне находится главное Меню, затем следуют Подменю, а на месте листьев находятся сообщения. При работе с подсистемой справки возможные ответы пользователя разбиваются на три группы: системные, ответы на Меню и Подменю. Системные включают команды: Н – начать, К – кончить, В – вернуться к предыдущему состоянию. Ответы на Меню содержат выбор пути по дереву. Ответы на Подменю включают: ВП n – вперед на n страниц, НЗ n – назад на n страниц, П – повторить первую страницу. Системные ответы могут быть использованы при работе в Меню и Подменю.

Подсистема решения предназначена для выбора задачи в пакете программ с возможностью получения справки по ней, формирования режимов решения, ввода и корректировки (с сохранением) исходных данных, генерации пускового пакета для ОС и постановки задания в очередь. Сценарий диалога должен обеспечить возврат в любое из указанных действий. Режимы решения можно выбирать по умолчанию или задавать в диалоге. Для построения диалога в этой подсистеме использованы вершины типа Меню (выбор задачи), Сообщение (работа с помощью), Таблица (формирование режимов работы, ввод исходных данных).

Для ввода данных предусматривается несколько возможностей: экран, карты, библиотека. Для каждой задачи имеется заготовка пускового пакета на ЯУЗ, содержащая переменные параметры. К ним относятся: имя программы (если их несколько для данной задачи), код пользователя, режимы работы, имя диска. Эти параметры могут заполняться в диалоге (вершина типа Таблица) или приниматься по умолчанию.

В системе ПРИМУС после помещения задания в очередь ОС логическая связь с ним разрывается. Поэтому разработана подсистема слежения, функции которой состоят в обработке системных очередей ОС, извлечении из них информации .о решаемой задаче и помещении этой информации в специальный НД. Диалог с этой подсистемой заключается в получении в любой момент времени информации о состоянии решения задачи: находится в очереди, обрабатывается процессором, имеются результаты.

После завершения выполнения задания пользователь может работать с подсистемой анализа. Ему предоставляются следующие возможности: просмотр протокола решения, распечатка протокола полностью или частично, сохранение протокола в базе данных. Каждый из этих подрежимов имеет свой сценарий диалога. При просмотре результатов на экране появляется первый фрагмент решения. Далее пользователю предоставляются следующие подкоманды: сдвиги вправо, влево, вперед и назад на заданное число позиций, сохранение фрагмента, поиск по контексту, подвод курсора к требуемой записи, получение подсправки о возможностях просмотра. Сценарии подсистемы анализа строятся с использованием вершин типа Меню, Процедура и Сообщение, Анализ.

Секции. Рассмотрим задачу построения процессов при обучении и контроле. Обучаемому предлагается ряд вопросов (меню), причем в зависимости от ответа определяется последовательность следующего вопроса. При обучении вместе с вопросом выдаются и возможные ответы. В зависимости от каждого ответа испытуемому ставится определенный балл, по окончании сценария контроля выдается оценка.

При проектировании диалоговой инструментальной системы контроля и обучения (ДИСКО) на базе монитора ДС ПРИМУС были использованы концепции, изложенные выше. Для достижения широкого класса ДС рассматриваемого типа достаточно трех системных переменных: Ответ–выбор в вершине типа меню; Балл – текущая оценка по результату ответа; Вершина – имя следующей вершины в зависимости от переменной Ответ. Сценарий будет включать вершины трех типов: Меню, Анализ и Процедура. Рассмотрим описание каждого типа вершин.

Меню состоит из трех секций: служебной, процедурной и секции экрана. При этом в секцию процедур входят возможные интервалы оценки и ответы. Вершина анализа включает служебную секцию и подсекцию интервалов. Вершина типа Процедура включает служебную секцию и подсекцию интервалов. Приведем структуру служебной секции:

[I= < количество интервалов в подсекции интервалов >]
[,V= < количество альтернатив в подсекции ответов >]
[, K= < команда перехода > | < оценка >]
[, H= < команда перехода на справочный поддиалог >]
[,O= < команда перехода на поддиалог ответа >]
[, M=X (в качестве ответа вводится номер; M=A (свободный ответ)]
[, T= < время существования системной транзитной фразы на экране >]
[, L= < имя подпрограммы выхода > (в вершину типа Программа)]

Подсекция интервалов включает: (A_1, B_1) < команда > ... (A_k, B_k) < команда >. Подсекция ответов включает: < возможный ответ > < команда > ... < возможный ответ > < команда >. При этом в качестве команды могут использоваться: + число | – число | = число | # число| команда перехода | транзитная реплика. Под числом понимается изменение текущего балла с + , – , = или без знака для специального случая выставления оценки. Команда перехода определяет дугу графа сценария. Под транзитной репликой понимается сообщение, которое на T секунд появляется на экране обучаемого после ввода им ответа и перед заданием очередного вопроса.

Команда перехода: {[ВАСК < имя вершины$_1$ >< имя вершины$_2$ >]| RETURN }, где ВАСК – шаг назад по дереву диалога; вершина$_1$ – начальная вершина вызываемого поддиалога; вершина$_2$ – вершина возврата в основной диалог (при отсутствии возврата по RETURN осуществляется переход в вызывающую вершину).

Программные и информационные средства. Для ввода и корректировки сценариев в отличие от других обучающих систем используется практически ограниченный естественный язык. При этом не требуется специальных инструментальных средств создания вершин по дереву сценария, поскольку для этого служит текстовый редактор системы ПРИМУС. В качестве базы для хране-

ния сценариев выбраны НД библиотечной организации с длиной записи 80 байт (наиболее распространенная длина строки терминала). Каждой вершине графа сценария ставятся в соответствие разделы библиотек. Таким образом, для проектирования информационных средств пользователь с экрана формирует по вышеописанным правилам вершины графа сценария контроля (обучения), предварительно продумав вопросы и оценки за них.

Важной особенностью системы ДИСКО является ее открытость, под которой понимается возможность подключения к вершине сценария прикладной или даже системной программы для выполнения нестандартных действий. Это позволяет задавать обучаемому вопросы, требующие сложного анализа ответа. Реализация сценария происходит путем интерпретации, что дает возможность проектировщику диалога обучения корректировать граф сценария и содержимое отдельных вершин прямо во время реализации диалогового процесса.

В состав программного обеспечения ДИСКО, оформленной как ФП системы ПРИМУС, входят следующие компоненты. *Блок реализации обменов* используется для выдачи вопросов, получения ответов обучаемого, выдачи транзитных сообщений и помощи. *Процедурный блок* изменяет текущую количественную оценку пользователя по результату очередного ответа, определяет следующую вершину в графе сценария и обращается к блоку загрузки. Блок загрузки на основании полученного идентификатора загружает требуемый раздел библиотеки сценария в оперативную память.

Для воспроизведения и разбора хода обучения или контроля *блок ведения истории диалогового процесса* позволяет преподавателю повторить путь в графе сценария. *Блок реализации взаимодействия* с прикладными программами подключает требуемые программы для анализа ответа обучаемого. *Блок отображения* позволяет преподавателю проконтролировать ход обучения (контроля) за каждым из терминалов.

Блок отладки позволяет проектировщику сценария обучения по каждой вершине вносить изменения путем работы с разделами библиотеки, включая служебные секции. Имеется возможность осуществлять принудительный переход к произвольной вершине, корректировать текущую количественную оценку, вводить другие вершины и выполнять ряд сервисных функций. Учитывая, что система реализует сценарии в режиме интерпретации, данные возможности могут осуществляться непосредственно в ходе выполнения разработанного ранее сценария.

Головной блок организует взаимодействие вышеописанных компонентов и сам вызывается как функциональная программа. Программное обеспечение системы не содержит специального блока помощи (HELP – процессор), что связано с наличием стандартных средств для его построения в каждом конкретном случае сценария, для чего введены возможные ссылки в произвольной вершине графа.

Система ДИСКО обеспечивает работу в следующих режимах: обучаемого, испытуемого, преподавателя и проектировщика. В *режиме обучаемого* предусмотрена возможность выдачи правильных ответов. В *режиме испытуемого* при ответе на вопрос возможен запрос помощи. *Режим преподавателя* позволяет оперативно вмешиваться в диалоговый процесс, следить за ходом процессов на всех терминалах, получать сводные данные о результатах контроля в

виде твердой копии (ведомости) на печатающем устройстве. В *режиме проектировщика* наряду с ранее рассмотренными возможностями имеется доступ к служебной информации сценария, дается доступ к корректировке сценария непосредственно в ходе диалогового процесса, а также предусматривается ряд отладочных подкоманд.

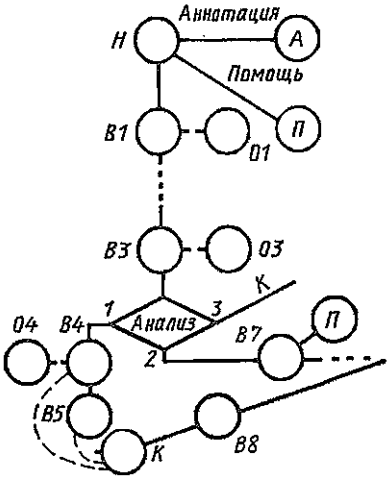


Рис. 6.4.
Сценарий диалога контролирующего типа.

Примеры проектирования диалоговых процессов. При проведении зачета предлагается сценарий, представленный на рис. 6.4. Его структура выбрана из следующих соображений. Первые три вопроса В1–В3 (можно и больше) задаются независимо от ответов и служат для предварительной оценки испытуемого, причем вопросы следует составлять в нарастающей сложности. По количеству набранных баллов за ответы на данные вопросы определяется дальнейший ход зачета: завершить с неудовлетворительной оценкой (до трех баллов); испытуемый претендует на отличную оценку (11–12 баллов); ответ колеблется между оценками "3" и "4" в остальных случаях. Различные варианты продолжения зачета определяются в секции интервалов вершины Анализ. По ветке *1* потенциальному отличнику задается один вопрос (В4), при правильном ответе на который он заканчивает зачет досрочно, в противном случае продолжает его (В5). В случае неверного ответа происходит переход в узел В8. При переходе на ветку *2* по вопросу В7 предусмотрена помощь. Все ветки сходятся в узле К, в котором выставляется конечная оценка. Программирование узла *1* выглядит следующим образом: V = 2, К = В2, М = Х; 1 + 3, +0. Это значит: в вершине выдаются два ответа, следующий вопрос В2, ответ сравнивается с номерами вопросов, за первый ответ начисляется 3 балла, за второй – 0. Для использования этого же сценария в качестве обучающего достаточно в каждом узле-вопросе добавить ссылку 0= ... на узел-ответ, а в последний узел ввести правильные аргументированные ответы. При этом в графе добавляются восемь (по числу вопросов) узлов-ответов.

Если в рассматриваемой системе зациклить вопрос до получения правильного ответа, то получится диалог игрового типа. Занятия в этом случае могут проходить по следующей схеме. Обучающиеся соревнуются, кто быстрее достигнет вершины-цели, для чего на экране появляется сообщение о ее достижении. При этом для стратегии ответа без размышления можно ввести транзитную решпику с большим T временем отображения на экране.

6.4. ПРОЕКТИРОВАНИЕ ДИАЛОГОВЫХ ПРОЦЕССОВ

ДС решения задач. При построении диалоговых взаимодействий с пакетами программ, которые были созданы для работы в пакетном режиме, возможны два варианта решения. В первом разработка диалоговых средств ведется под конкретный пакет программ, решающий ограниченный класс задач. Во втором случае диалоговые средства являются почти универсальными, использование которых возможно с достаточно широким классом прикладных программ. При первом подходе диалоговый язык весьма жестко ограничивается

кругом задач, решаемых прикладными программами. Во втором случае языковые средства диалога позволяют создавать сценарии с различными пакетами. Охарактеризуем более подробно реализацию этих двух подходов.

Рассмотренная ДС решения и анализа, структура которой приведена на рис. 6.3, относится к проектированию диалоговых процессов первого типа. Из четырех подсистем две включают средства настройки на пакет программ: справочная и слежения. Для проектирования процессов получения справки по проблемной ориентации пакета достаточно сформировать сценарий средствами текстового редактора. Подсистема работает с программным модулем чтения раздела библиотеки, содержимое которого отображается на экране. Подсистема слежения использует модуль определения существования заданного раздела библиотеки, который содержит информацию о заданиях.

Подсистема решения настроена на конкретный пакет программ, поскольку ее сценарий включает вершины выбора задачи, формирования режимов решения, ввода данных и формирования пускового пакета. Для этого используются как стандартные программы системы ПРИМУС для копирования НД, корректировки данных, а также запуска заданий под управлением ОС, так и нестандартные. Последние предназначены для генерации пускового пакета, формирования DD-карт, работы с разделами библиотек. Подсистема решения использует НД спецификаций пакета программ, в котором записаны виды решаемых задач, их характеристики, вводимые в диалоге параметры, параметры, задаваемые по умолчанию. Для заготовок пусковых пакетов на ЯУЗ имеется НД, который в диалоге заполняется для каждой конкретной задачи рядом параметров, производится их контроль, а затем запускается сгенерированное задание. Спецификации задач и заготовки пусковых пакетов являются элементами диалоговой технологии решения задач.

Подсистема анализа работает с модулями просмотра результатов, печати их на АЦПУ и сохранения в базе данных. Эти модули реализуют сценарии, рассмотренные в § 6.2. Для своей работы модули используют НД для временного хранения результатов, НД виртуального печатающего устройства и НД для вывода на перфоленту.

ДС проектирования сценариев. Для реализации второго подхода предложена трехступенчатая схема организации диалога с пакетом программ. На уровне пользователя реализуются сценарии, которые описываются языковыми средствами, рассмотренными на примере системы ДИСКО. На логическом уровне обрабатываются структуры сценариев средствами некоторой универсальной диалоговой системы, независимо от содержания обменов с прикладными программами. На физическом уровне хранятся тексты обменов сценария, которые поддерживаются и обрабатываются модулями, работающими с библиотечными НД.

В качестве данной концепции рассмотрим построение диалоговой инструментальной системы проектирования сценариев (ДИСПРОС), которая может быть реализована в среде монитора ПРИМУС. ДИСПРОС является надстройкой над пакетами прикладных программ, не имеющих собственных диалоговых средств. Система обеспечивает следующие функции: интерактивный сбор информации, необходимый для запуска ПП, и их выполнение; анализ результатов работы ПП; организацию справочных диалогов; запоминание пути прохождения по графу сценария для повторного решения задачи.

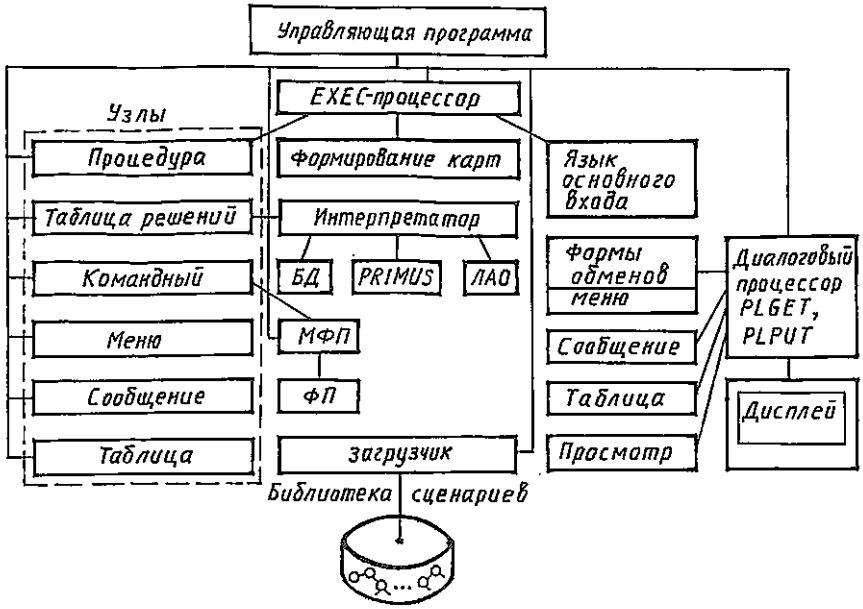


Рис. 6.5.
Структура диалоговой инструментальной системы проектирования сценариев.

Входной информацией для ДИСПРОС служат граф-сценарии, описывающие процесс решения задачи в данном пакете ПП. При этом отдельные вершины графа являются инструкциями для ДС на выполнение определенных функций: выбор альтернативы для определения дальнейшего пути решения (автоматически или на основании решения пользователя); ввод данных, необходимых для работы прикладных программ; запуск ПП и анализ результатов их работы; обработка других ситуаций, предусмотренных сценарием. Основное назначение системы – переход к эксплуатации в диалоговом режиме широкого набора пакетов ПП, начиная от задач вычислительного характера и кончая утилитами ОС ЕС.

Общая последовательность работы по проектированию диалоговых процессов с ПП, которые обрабатывались в пакетном режиме, заключается в следующем. Разрабатывается сценарий диалога полностью на естественном языке в терминах обменов и связей между ними. Затем, используя языковые средства описания вершин, производится программирование обменов (на ограниченном естественном языке), как и для системы ДИСКО. Далее проектировщик с терминала средствами текстового редактора системы ПРИМУС (если проектирование ведется на ЕС ЭВМ) или редактора на другой машине создает библиотеку сценариев, записывая тексты обменов в разделы. Наконец, запускается система ДИСПРОС, в качестве параметров которой передаются имена библиотеки сценария, и начинается диалоговая работа с ПП.

Программно-информационное обеспечение ДИСПРОС. Структура системы приведена на рис. 6.5 и включает следующие компоненты: управляющую про-

грамму, диалоговый процессор, исполняющий (EXEC) процессор, интерпретатор, загрузчик сценариев, монитор ФП.

Диалоговый процессор реализует взаимодействие с пользователем (на основании программ базового монитора ПРИМУС PLENV, PLGET, PLPUT). Он реализует основные виды обменов: меню, сообщение, таблица, просмотр результатов (анализ) и др.

Загрузчик элементов сценария выбирает по дереву очередную вершину, производит загрузку в оперативную память соответствующего раздела, используя библиотечный метод доступа BPAM.

Интерпретатор выполняет обработку логико-арифметических операций (ЛАО) в секции процедур, работу с базой данных в случае запроса сложной формы, а также обработку выражений, включающих обращение к командам системы ПРИМУС. Для реализации сложной логики переходов интерпретатор обрабатывает таблицу решений, в которой кодируется алгоритм управления прикладными или системными модулями.

Монитор ФП интерпретирует вершину командного типа и является составной частью системы ПРИМУС.

Исполняющий процессор реализует вершины типа Процедура, организует запуск ПП под управлением ОС. Он использует интерпретатор DD-предложений, построенный на базе ФП ПРИМУС FPOODLST , а также работает с языком основного входа. Последний предназначен для генерирования пусковых пакетов ОС, как и в ДС решения задач, а также для описания параметров работы с программами (ввод данных), включая утилиты.

Кроме основных блоков, в состав ДИСПРОС входят блок анализа результатов решения ПП и блок сохранения истории диалоговых процессов. Информационное обеспечение системы включает библиотеки сценариев, ПП, НД ввода и вывода, служебные, истории диалога.

6.5. ПОДХОД К ДИАЛОГОВОМУ КОНСТРУИРОВАНИЮ ЗАДАЧ

Постановка проблемы. Потребность в квалифицированных программистах становится сейчас все более острой, поскольку в эксплуатацию вводится все большее число мини- и микроЭВМ. Решению этой проблемы могли бы помочь наборы стандартизованных модулей, работающих под управлением ОС, но для адаптации этих программ к требованиям пользователя необходима работа программиста. Выходом из этой ситуации является переложение функций программиста на специальную систему автоматизации конструирования задач. Методы и средства, реализованные в таких системах, могут быть совершенно различными. В одних используются непроцедурные языки для описания задачи, которые настолько просты в изучении, что непрограммист может быстро освоить работу с ними. Другие системы расширяют версии систем управления БД, оснащенные средствами для создания законченных ПП.

В настоящее время пакеты автоматизации программирования решают прикладную задачу одним из двух способов: либо генерируют программу на машинном языке, либо составляют набор заранее отлаженных модулей на время выполнения прикладной задачи. К сожалению, такие пакеты ориентированы в основном на обработку задач экономического типа.

Представление структур задач. При решении задач проектирования и уп-

равления имеются библиотеки готовых модулей, из которых необходимо конструировать задачи для тех или иных приложений. На организацию модулей следует наложить ряд ограничений: свести их в централизованную базу модулей, ввести элементы их спецификаций. Под последними будем понимать описание знаний о задаче на естественном языке, описание требуемых входных данных, ресурсов, выходных данных, т. е. "паспорта". Для работы модулей нужны входные и выходные НД, для которых тоже необходимы соответствующие спецификации.

Для автоматизированной обработки задач вводятся три уровня описания. На *уровне пользователя* задача описывается на естественном языке (справочные данные) и в диалоговом режиме по запросам системы определяется ряд параметров (требуемые входные НД, режимы работы, генерация управляющих операторов ОС). При этом может использоваться язык описания модулей. На *внутреннем (логическом) уровне* программа конструирования, обрабатывая полученные на внешнем уровне данные, строит требуемую последовательность модулей, проверяет для них готовность входных НД, вызывает эти модули для выполнения (динамически как подзадачи или путем постановки в очередь), заполняет выходные наборы данных, осуществляет (автоматически или в диалоге с пользователем) логику переходов в задаче. На *физическом уровне* средствами ОС выполняется работа с модулями.

Модель построения задач. Для реализации рассмотренных выше принципов необходимо иметь модель, позволяющую универсальным образом обрабатывать программные модули. Пусть в состав i задачи, решаемой пакетом, входят n модулей, определенным образом связанных между собой по обмену информацией. Используем для описания структуры задачи графовую модель, рассмотренную выше.

Опишем структуру решаемой задачи направленным графом $\Gamma = (V, E)$, где V и E – множество вершин и дуг. Поставим в соответствие множеству вершин $V = \{v_i | i = \overline{1, n}\}$ множество модулей, образующих задачу, а множеству дуг – те информационно-логические связи, в которые вступают между собой модули в процессе решения задачи. Введем множество $S = \{s_j | j = \overline{1,k}\}$, элементы которого s_i позволяют однозначно осуществить переход как в вершину v_i, так и из нее. Введем множество C логических условий $C = \{c_j | j = \overline{1,n}\}$, которые вырабатываются модулями, соответствующими вершинам v_j при переходе к вершинам v_i.

Рассмотрим последовательность обработки графа. Пусть имеется вершина v_j, из которой нужно совершить переход. Этой вершине соответствует вырабатываемое условие c_j, определяющее однозначность перехода в вершину v'_k. В автоматическом режиме по условию c_j для каждой вершины v_j переход в v'_k определяется системой, в диалоговом вершина v'_k выбирается оператором. В последнем случае проверяется условие допустимости перехода. Для вершины v_j и смежной с ней вершины v'_k переход будет существовать тогда, когда выполняется условие $c_j = s_j \cap s_k$. Вначале проверяется правильность выработанного условия перехода $c_j \in C$; если $c_j \in C$, следовательно, условие неверно и выполняются следующие действия. В автоматическом режиме программа, соответствующая вершине v_j, снова просчитывается, в диалоговом режиме пользователю выдается сообщение о неправильно выработанном c_j.

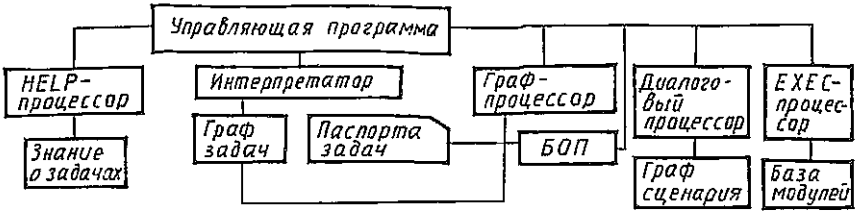


Рис. 6.6.
Структура диалоговой инструментальной системы конструирования задач.

Дальнейшая работа зависит от его решения. В случае выполнения условия перехода определяется вершина v_k . Для модуля, соответствующего этой вершине, проверяется готовность входных НД, в случае их готовности управление передается данному модулю. Если НД для модуля, соответствующего вершине v_j , не готовы, проверяется причина. При необходимости управление может быть передано модулям, заполняющим НД для модуля, соответствующего вершине v_j . Если вершины v_k не находится, следовательно, вершина v_j была конечной, и обработка графа Г на этом заканчивается.

Структура системы конструирования задач. Предложенная модель управления позволяет организовать следующие запросы: ВЫПОЛНИТЬ ЗАДАЧУ параметры, ПОЛУЧИТЬ ФАЙЛ параметры, ВОЗМОЖНЫЕ ДЕЙСТВИЯ ПОЛЬЗОВАТЕЛЯ. В первом случае проверяется возможность генерации цепочки модулей, в диалоге могут запрашиваться дополнительные данные. При втором запросе проверяется, какие задачи дают на выходе требуемый файл (для них генерируются соответствующие цепочки). В третьем случае пользователю выдается информация о возможных решаемых задачах.

К другим функциям управления конструированием задач относятся: выдача справочных данных, поддержание диалога с пользователем, организация информационного обмена между модулями, обработка и корректировка служебной информации о структурах задач, запуск модулей на выполнение. Для реализации этих функций разработана диалоговая инструментальная система конструирования задач (ДИСКОЗ).

В состав системы входят следующие компоненты: управляющая программа, подсистема справки, диалоговый процессор, инициирующий процессор, граф-процессор, блок обработки паспортов (БОП), интерпретатор языка описания структур задач (рис. 6.6). Подсистема справки позволяет проектировщику познакомиться с возможностями ПП, практически работая с описаниями задач. Диалоговый процессор осуществляет все обмены в системе. Интерпретатор преобразует описание задачи пользователем во внутреннее представление, которое обрабатывается граф-процессором. Для обработки системной информации о модулях (паспортах) используется БОП. Для запуска выбранного модуля служит инициирующий процессор. Информационное обеспечение системы включает наборы паспортов модулей, описания структур задач, описание НД задач, сценарии диалога, служебные НД для привязки к ОС и ПРИМУС, а также базу прикладных модулей.

Выводы

1. Диалоговым расширением базовой ОС ЕС является открытая система ПРИМУС, обеспечивающая многотерминальную работу пользователей. На логическом уровне организацию процессов осуществляет ядро, включающее монитор программных процессов, планировщик, монитор команд, ПУС для связи с терминалами и ПУВ для вывода на печать. На пользовательском уровне имеется набор команд, поддерживаемый функциональными командами монитора команд. Система работает как одношаговое задание ОС ЕС.

2. Вычислительные процессы в диалоговой системе с перестраиваемой структурой организуются на базе хранимых в памяти сценариев. Последние могут представляться многодольным графом с вершинами типа: Меню, Таблица, Сообщения, Программа, Процедура и Анализ. Для организации диалогового процесса достаточно средствами текстового редактора создать граф сценария, поместить его в библиотечный набор, а затем интерпретировать его написанным монитором обработки вершин сценария.

3. Диалоговая система контроля и обучения может включать вершины четырех типов: Вопрос – для формирования вопросов и возможных ответов, Ответ – для подсказки, Анализ – для разветвления по графу в зависимости от набранных баллов и Оценка – для выдачи результата контроля. Программирование сценария обучения выполняется на ограниченном естественном языке под управлением диалогового монитора, например ДС ПРИМУС.

4. Диалоговая система проектирования сценариев с пакетами программ может включать обработку следующих вершин: Меню – для выбора действий, Программа – для запуска прикладной программы, Процедура – для выполнения системной программы, Анализ – для просмотра результатов. Под конкретный пакет в диалоге формируется требуемый сценарий, который затем интерпретируется.

5. Диалоговая система конструирования задач из готовых модулей интерпретирует элементы знаний о структурах задач, составе "паспортов" модулей и наборах данных. Система может реализовывать запросы: ВЫПОЛНИТЬ ЗАДАЧУ, ПОЛУЧИТЬ ФАЙЛ, ВОЗМОЖНЫЕ ДЕЙСТВИЯ ПОЛЬЗОВАТЕЛЯ.

Рекомендуемая литература: [7, 16].

Вопросы для контроля

1. Поясните синхронизацию процессов в ДС ПРИМУС.
2. Как разрабатываются пользовательские процессы в ДС ПРИМУС?
3. Как строятся сценарии диалога?
4. Охарактеризуйте возможные типы секций вершины графа диалога.
5. Каким образом можно разработать сценарий контроля?
6. Поясните пути проектирования диалоговых процессов с ПП.
7. Поясните графовую модель обработки задачи.

7. МНОГОМАШИННАЯ И МУЛЬТИПРОЦЕССОРНАЯ ОБРАБОТКА ИНФОРМАЦИИ

7.1. ОРГАНИЗАЦИЯ МНОГОМАШИННЫХ И МУЛЬТИПРОЦЕССОРНЫХ СИСТЕМ

Общие сведения. В отличие от организации вычислительных процессов в однопроцессорных ЭВМ ОВП в многопроцессорных и многомашинных ВС имеют ряд особенностей: 1) в планировании и синхронизации параллельных вычислительных процессов; 2) организации межмашинного, межпроцессорного обмена; 3) распараллеливании задач на различных уровнях: языка, компилятора, ОС и аппаратурном; 4) диспетчеризации задач, особенно связанных.

Программные средства для работы в параллельных ВС имеются на уровне ОС (реализация ряда дополнительных функций ядра, а также планировщики для организации очередей к ресурсам и диспетчеры для обслуживания этих очередей), компиляторов с параллельных языков программирования и языковых средств распараллеливания вычислений. С 70-х годов разрабатываются ВС, в которых широко используется параллельная обработка информации в двух направлениях. Первое направление – создание многомашинных ВС, в основе которых лежит объединение информационно- и программно-совместимых ЭВМ. Второе – создание многопроцессорных ВС (МВС), включающих сеть центральных, периферийных процессоров с общей или разделяемой памятью. В дальнейшем под МВС будем понимать систему с двумя и более процессорами обработки информации, основной памятью, периферийными процессорами и внешними устройствами, работающими под управлением одной ОС.

Различные МВС могут быть ориентированы на следующие виды параллелизма: множество объектов (естественный параллелизм), множество задач, независимых ветвей задач, смежных операций.

Признаки классификации вычислительных систем. В качестве основных признаков классификации, характеризующей организацию структуры и функционирования ВС с точки зрения параллельности, используются следующие: тип потока команд в центральной части ВС – одиночный (ОК) и множественный (МК), тип потока данных в центральной части ВС – одиночный (ОД) и множественный (МД); степень связанности компонентов системы (низкая и высокая); степень однородности основных компонентов системы – однородные и неоднородные ВС; тип внутренних связей в системе (типа канал – канал, через внешнюю память, через оперативную память и непосредственно между процессорами). Возможна также организация систем со связями через общую шину с разделением во времени, через множество шин при использовании многовходовой основной памяти и с перекрестными связями при помощи матричного коммутатора. Наиболее сложную структуру с точки зрения приведенной классификации имеют многомашинные и многопроцессорные ВС.

Характерными типами связей в многомашинных системах являются связи канал–канал посредством адаптера, через внешнюю память и непосредственно между процессорами для обмена сигналами и прямого управления.

Одиночные и множественные потоки команд и данных. Понятия основных ВС по этим признакам классификации были предложены Д.Флином. В соответствии с этим введено четыре класса систем. В системах типа ОКОД реализуется обычная однопроцессорная ЭВМ. При этом под потоком команд понимается ряд команд, выполняемых системой, а под потоком данных – ряд данных, вызываемых потоком команд, включая промежуточные результаты.

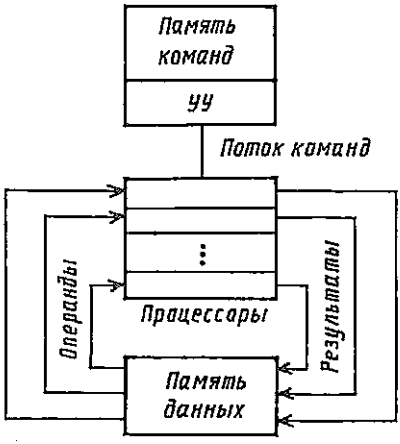


Рис. 7.1.
Структура ВС типа ОКМД.

В системе типа ОКМД (рис. 7.1) одно устройство управления (УУ) организовывает работу множества процессорных модулей так, что каждый из них выполняет одну команду над различными данными в конкретный момент времени. При этом часть процессоров может пропускать конкретные команды, что определяется операциями маскирования. В ОКМД реальная скорость обработки сильно зависит от возможностей загрузки процессорных модулей путем распараллеливания вычислений. Данные многопроцессорные системы получили название *матричных.*

В системе типа МКОД вычислительный процесс разбивается на несколько этапов, каждому из которых соответствует один из процессоров (рис. 7.2). Эти модули в совокупности составляют магистраль обработки (конвейер процессоров). Реальная скорость обработки зависит от возможности заполнения магистрали. Высокая скорость достигается при выполнении длинных линейных участков программ. При частых прерываниях линейных программ скорость обработки снижается. Данные ВС получили название *конвейерных.*

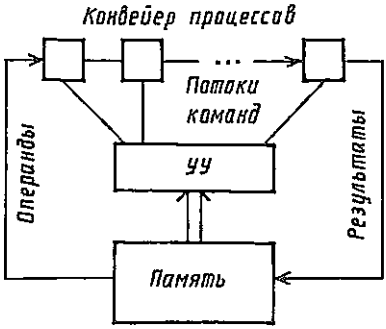


Рис. 7.2.
Структура ВС типа МКОД.

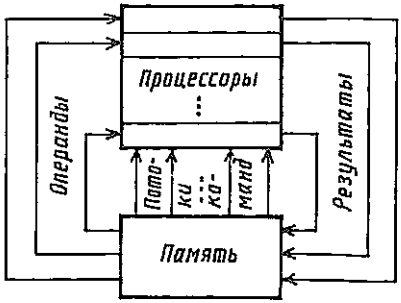


Рис. 7.3.
Структура ВС типа МКМД.

В системах типа МКМД несколько управляющих устройств осуществляют управление одновременным выполнением различных участков одной и той же программы (рис. 7.3). Реальная скорость обработки информации зависит от возможности загрузки процессоров; если затруднено распараллеливание одной программы, то можно выполнять несколько программ. Эти системы отличаются высокой надежностью благодаря возможности резервирования однотипных устройств.

Однако схема четырех классов обладает недостатками, поскольку структурно различные матричные и ассоциативные системы попадают в один класс ОКМД. В целях улучшения схемы ее модифицируют для обеспечения различия между пословной и поразрядной обработкой. Это приводит к Эрланской схеме классификации (ЭСК), которая базируется на триплете, определяемом так: $f = (k, d, w)$, где k – число устройств управления (УУ); d – число обрабатывающих устройств (АЛУ), управляемых одним из k устройств; w – число разрядов, обрабатываемых каждым из d АЛУ. Однопроцессорная ЭВМ по этой классификации характеризуется триплетом $f = (1, 1, w)$, где в различных моделях $w = 8, 16, 32, 64$ разряда.

Введенные базисные триплеты позволяют описывать системы, имеющие либо несколько однородных УУ, либо несколько АЛУ, каждое из которых может обрабатывать один поток данных. Если в системе имеется несколько различных элементов, необходима комбинация триплетов. Для этого введены операции: + – для систем, содержащих две и более различные структуры; * – для систем с последовательно упорядоченными структурами, в которых данные обрабатываются всеми структурами. Для разъяснения функций операторов + и * рассмотрим пример. В системе CDC 7600 пять 12-разрядных слов из периферийных процессоров передаются в одно 60-разрядное слово центрального процессора. Тогда имеем описание ВС $f_{CDC} = (5, 1, 12) * (1^*, 9, 60)$. В первом члене описывается 5 процессоров, которые взаимодействуют с одним 9-конвейерным процессором. С помощью данной схемы можно классифицировать практически все сложные составные структуры ВС.

Структура связей в параллельных ВС. Если рассмотреть связи для обмена информацией между процессорами на уровне оборудования, то схема классификации может включать 4 уровня. *Первый уровень* определяет стратегию организации обмена информацией и включает прямые (передатчик–приемник) и косвенные (между приемником и передатчиком есть промежуточные пункты) тракты передачи. *Второй уровень* определяет метод соединения трактов обмена, которые могут быть централизованными и децентрализованными. Косвенная передача в первом случае связана с единым промежуточным пунктом между приемником и источником. Косвенная передача при децентрализованном управлении связана с выбором одного из нескольких альтернативных пунктов передачи информации.

Третий уровень соответствует структуре путей передачи с индивидуальными и общими (разделяемыми) трактами обменов. *Четвертый уровень* включает различные типы структур ВС (рис. 7.4). Прямой передаче по разделенным путям соответствует кольцевая структура процессорных элементов (рис. 7.4, *а*), непрямой с набором связей – лучевая структура (рис. 7.4, *в*), прямой передаче с общим путем – многопроцессорная структура с основной памятью (рис. 7.4, *б*) и с общей шиной (рис. 7.4, *г*). Непрямой пе-

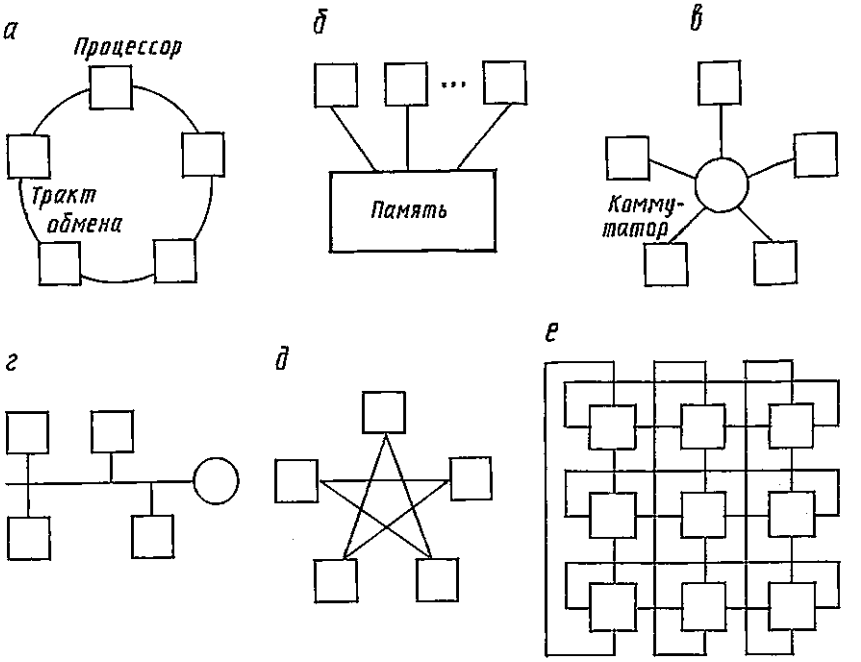


Рис. 7.4. Виды межпроцессорных связей в мультипроцессорных ВС.

редаче по раздельным путям при централизованном управлении соответствует структура звезды (рис. 7.4, *д*), непрямой передаче по раздельным трактам при децентрализованном управлении – регулярная структура (рис. 7.4, *е*), при которой каждый процессор связан с четырьмя соседними.

7.2. ОРГАНИЗАЦИЯ РАБОТЫ В МУЛЬТИПРОЦЕССОРНЫХ И МНОГОМАШИННЫХ СИСТЕМАХ

Коммутация потоков команд и данных. В ВС типа ОКМД с единственной памятью программ и несколькими модулями памяти данных коммутируется только один поток команд, распределенный между соответствующими устройствами обработки (рис. 7.5). На схеме обозначены: M_1 – память программы; M_2 – память данных; S_3 – коммутаторы; P_4 – устройства обработки.

В ВС типа МКМД нередко используются общая память и несколько процессоров, которые имеют локальную память команд и данных. Общая память коммутируется с каждым процессором (рис. 7.6). Для исключения конфликтов могут применяться несколько копий одной и той же программы. Здесь введены обозначения M_1 – общая память; M_2 – локальные памяти; S_3 – коммутаторы; P_4 – процессоры. Каждый блок может загружать программу (сегмент) в свою локальную память и работать автономно. Коммутаторы осуществляют внешние по отношению к блокам обработки функции переключения связей между этими блоками и внешней памятью. Однако может быть организована прямая связь между блоками обработки без внешней памяти.

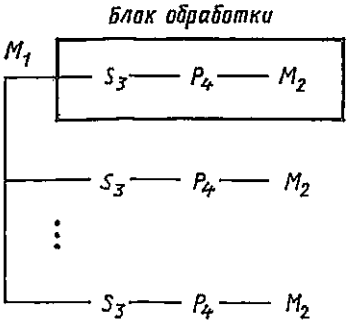


Рис. 7.5.
Коммутация потоков команд.

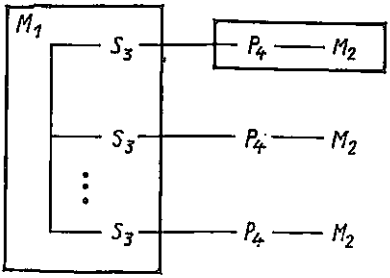


Рис. 7.6.
Коммутация блоков обработки.

Параллельные системы на базе ЕС ЭВМ. При разработке технических средств и ПО ЕС ЭВМ предусмотрено комплексирование на уровне процессоров, каналов, внешней памяти (многомашинные ВС) и основной памяти (многопроцессорные ВС).

Рассмотрим средства комплексирования на каждом из указанных уровней. Для обмена управляющей и синхронизирующей информациями между процессорами ВС используются средства прямого управления, к которым относятся команды ПРЯМАЯ ЗАПИСЬ и ПРЯМОЕ ЧТЕНИЕ. Один из процессоров может связываться с другим посредством этих команд и механизма внешних прерываний. Физическая связь осуществляется по кабелям интерфейса прямого управления, к которому может быть подключен пульт управления комплексом, позволяющий изменять режимы работы.

При комплексировании на уровне каналов используется адаптер, имеющий выходы на стандартный интерфейс ввода-вывода и подключающийся к каналам ЭВМ. Скорость обмена через адаптер близка к скорости работы канала. При этом в обмене участвует не только управляющая информация, но и массивы данных.

При комплексировании на уровне внешней памяти используются двухканальные переключатели, позволяющие подключить устройство управления НМЛ и НМД к двум каналам различных ЭВМ. В результате образуется общее поле памяти на управляемых накопителях. Для согласования работы с общим полем внешней памяти служат команды двухканальных переключателей, обеспечивающие резервирование устройства во время работы канала и освобождение его после завершения работы. Наиболее гибкой и быстрой является связь процессоров через общее поле основной памяти, которое организуется по принципу многошинной многовходовой памяти.

Одной из последних моделей ЕС ЭВМ является ЕС 1065 (1066), выполненная в виде мультисистемы для работы в мультипроцессорном и многомашинном режимах. Максимальная конфигурация мультипроцессора включает четыре процессора команд, два устройства управления памятью и два операционных устройства. Каждый процессор команд реализует одну программу и имеет четыре уровня совмещения выполнения команд. Операционные устройства осуществляют весь набор операций ЕС ЭВМ на двух уровнях совмещения. Уст-

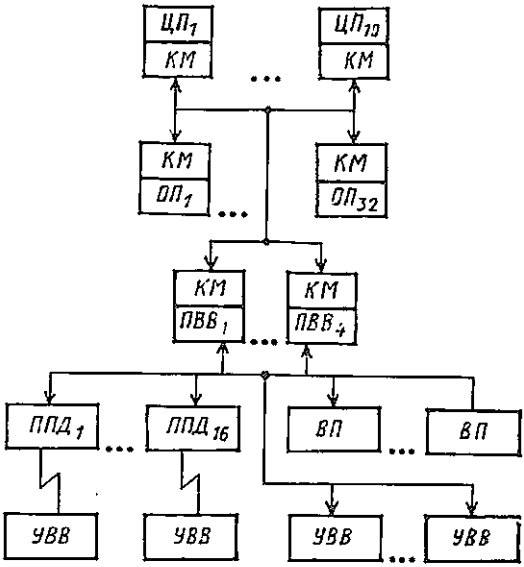


Рис. 7.7.
Структура системы "Эльбрус".

ройства управления памятью обслуживают процессоры команд и мультиплексоры (4 канала) ввода-вывода. В состав ОС ЕС 1065 включено программное обеспечение средств комплексирования для управления техническими средствами.

Функционирование многопроцессорной системы. Работу такой системы рассмотрим на примере комплекса "Эльбрус", структура которого представлена на рис. 7.7. Многопроцессорный вычислительный комплекс (МВК) "Эльбрус" содержит модули центральных процессоров ($ЦП_1$–$ЦП_{10}$), основной памяти ($ОП_1$–$ОП_{32}$), процессоров ввода-вывода ($ПВВ_1$–$ПВВ_4$), процессоров приема-передачи данных ($ППД_1$–$ППД_{16}$), а также внешней памяти (ВП) и устройств ввода-вывода (УВВ). Модули процессоров и памяти связаны между собой с помощью центрального коммутатора (КМ), модули которого разнесены по функциональным модулям. Коммутатор соединяет четыре модуля ОП со всеми модулями процессоров центральных и ввода-вывода. Процессоры приема-передачи, промежуточная и внешняя память, устройства ввода-вывода подключаются к центральной части через процессоры ввода-вывода. Линии связи подсоединяются к процессорам приема-передачи через адаптеры и групповые устройства сопряжения.

Модули системы работают параллельно и независимо друг от друга, ресурсы системы динамически распределяются ОС. В системе осуществляется резервирование на уровне функциональных модулей. Неисправный модуль выводится из работы, а его функции передаются другим модулям такого же типа.

В основу организации вычислений положен стек, применяемый как для вычисления выражений, так и для динамического распределения памяти под

процедуры. Каждое слово имеет описатель тег, указывающий тип данных и формат (32, 64, 128 разрядов). Адресация осуществляется с помощью адресной пары в виде кода процедуры, определяющего базу, и номера слова в стеке данных этой процедуры. Коды команд загружаются в физическую память, а данные – в виртуальную. Обрабатываемые данные представляются массивами и описываются дескрипторами.

Вычислительный процесс может быть активным и пассивным. В первом случае он отображается стеком, базовыми регистрами, сверхоперативной памятью и другим оборудованием ЦП. Во втором случае процесс ожидает события и отображается данными в памяти. Для распараллеливания вычислений под каждую процедуру отводится свой стек. При этом любой процессор может работать на любом стеке. Для синхронизации процессов используются операции ОТКРЫТЬ СЕМАФОР и ЗАКРЫТЬ СЕМАФОР. При работе системы готовые к выполнению процессы находятся в очереди, а освобождающиеся процессоры средствами диспетчера ОС назначаются для ее обслуживания.

Пусть все процессоры заняты работой и имеются также готовые к выполнению процессы в очереди, ожидающие освобождения хотя бы одного из процессоров. Как только некоторый активный процесс переходит по той или иной причине в пассивный или заканчивается, соответствующий процессор освобождается и обращается к очереди готовых процессов. Ему назначается диспетчером процесс, который начинает выполняться на данном процессоре. Процессы в очереди могут распределяться в зависимости от приоритетов планировщиком. Диспетчеризация и планирование работы МВК осуществляется средствами ОС.

Сверхоперативная память распределена по процессорам и разбита в каждом из них по функциональным признакам на виртуальные быстрые регистры для буферизации вычислений выражений (верхушка стека), прямореализуемую память для буферизации локальных данных процесса (линейный участок стека), ассоциативную память для буферизации команд, массивов и глобальных данных процесса. Буфер команд обеспечивает опережающую их выборку, буфер данных массивов – опережающую подкачку данных.

При трансляции программ однородная информация объединяется в сегменты, которые описываются дескрипторами. Объем выделяемой физической памяти определяется этими дескрипторами. Организация вычислений на базе стеков позволяет вызывать данные в физическую память по мере необходимости и освобождать ее по завершении вычислений. Перечисленные методы повышают эффективность использования основной памяти, но усложняют ОС и доступ процессора к данным, что компенсируется введением сверхоперативной памяти.

Главные функции обмена между основной памятью и периферийными устройствами и парами периферийных устройств выполняет процессор ввода-вывода. Он обрабатывает список заявок на ввод-вывод, который создается центральными процессорами. Выполняя очередную заявку, процессор осуществляет запуск периферийного устройства, реализует передачу данных, завершает обмен и корректирует список заявок. Если существует несколько путей обмена, то происходит выбор пути по мере освобождения оборудования. Все функции управления внешним оборудованием, включая диспетчеризацию, резервирование и выбор канала, реализуются аппаратно. Это позволяет разгру-

жать центральные процессоры от прерываний ввода-вывода.

В процессорах приема-передачи данных принят программный способ взаимодействия с объектом из-за их многообразия. При этом для подключения нового типа объекта требуется создание очередного программного модуля с описанием этого объекта.

В состав программного обеспечения МВК "Эльбрус" входят система программирования ЭЛЬ-76, единая ОС, система стандартных и сервисных программ, система телеобработки, тестовые и диагностические программы.

Система программирования предоставляет пользователю языки: АЛГОЛ, ФОРТРАН, КОБОЛ, ПЛ/1, СИМУЛА, ПАСКАЛЬ и АВТОКОД. Она обеспечивает отладку на уровне исходных программ, диагностику ошибок при компиляции и исполнении, комплексирование и сегментацию программ, редактирование.

Операционная система обеспечивает мультипрограммные режимы ближней и дальней пакетной обработки, режим РВ и терминальную обработку. Она осуществляет синхронизацию процессоров, а также реализует динамическое распределение всех ресурсов системы и поддерживает виртуальную память. ОС обеспечивает также автоматическую реконфигурацию МВК, восстановление файлов и перезапуск задач и системы в целях ее функциональной надежности.

Программное обеспечение МВК "Эльбрус-1" и "Эльбрус-2" совместимо. В системе ввода-вывода аппаратно реализованы основные алгоритмы диспетчера ОС и управления работой периферийных устройств.

При разработке программного обеспечения создан основной набор функций, которые в сочетании с гибкими средствами комплексирования позволяют эффективно строить программы. ВС с ее программным обеспечением как бы адаптируется к решаемым задачам, а не наоборот, когда из-за негибкости системы необходимо модифицировать алгоритм.

Программное обеспечение комплекса строится с учетом сближения системного и прикладного программирования. В обоих случаях программирование ведется на языке высокого уровня, почти все механизмы для пользователей реализованы программно.

Модель комплекса "Эльбрус-1" включает до 10 центральных процессоров, 32 модулей основной памяти, 4 процессоров ввода-вывода и 16 процессоров приема-передачи. Имеется возможность выполнения программ БЭСМ-6 применительно к системе Диспак с подключением процессора, реализующего систему команд БЭСМ-6.

МВК ПС-3000. Особенностью структуры данного МВК является то, что разработан лишь узел центральной обработки, а в качестве периферийного оборудования используется оборудование систем ЕС и СМ. В узел центральной обработки входят четыре скалярных и два векторных процессора, оперативная память и спецпроцессор ПС-2000.

Скалярный процессор подготавливает и обрабатывает данные и управляет векторным процессором (один управляет, второй подготавливает данные). *Векторный процессор* производит операции над векторами. Спецпроцессор ПС-2000 является матричным со структурой МДОК. ОС реализована в узле центральной обработки для разгрузки векторных процессоров от прерываний. Часть функций ОС возложена на периферийные процессоры.

7.3. ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ И ПАРАЛЛЕЛЬНЫЕ ПРОЦЕССЫ

Основные понятия. Задачи расчета характеристик и планирования параллельных вычислительных процессов (ВПр) объединяются в рамках проблем параллельного программирования. При их решении используются специализированные модели параллельных и, как частный случай, последовательных программ. В этих моделях основное внимание уделяется связям по управлению без учета смысловых характеристик. Задачи параллельного программирования могут быть классифицированы следующим образом (рис. 7.8): построение моделей параллельных программ (А), расчет характеристик параллельных вычислительных процессов (В) и планирование параллельных вычислительных характеристик (С). В свою очередь модели параллельных программ разделяются на матричные *1* и графовые *2*. Расчет характеристик параллельных ВПр производится на основе методов линейной алгебры *3* и преобразования графов *4*. Планирование параллельных ВПр разделяется на статическое *5* и динамическое *6* (составление расписаний).

При разработке моделей параллельных программ используются ориентированные графы с конечным числом вершин и дуг. Вершины графа ставятся в соответствие участкам (сегментам, модулям) программ, а дуги – связям между этими участками. Параметры графов могут быть как вероятностными, так и детерминированными. В простых моделях графы не имеют циклов и ветвлений, в более реальных моделях содержат циклы и (или) ветвления. С вершинами сопоставляются время выполнения модулей или затраты памяти. Дугам ставятся в соответствие логические или вероятностные характеристики переходов между модулями программы.

Модели программ могут быть представлены в матричной форме, когда элементы матриц могут показывать связи между вершинами графов. При расчете операционных характеристик параллельных процессов применяются методы линейной алгебры для матричных моделей и методы преобразования графов. В первом случае можно вычислить оценки любых характеристик, во втором – лишь оценки общих характеристик, но для больших моделей. Модели параллельных программ и характеристики процессов служат для планирования процессов, методы которых будут рассмотрены в § 7.4.

Программные системы реального времени. Построение моделей, расчет характеристик и построение расписаний необходимы в первую очередь при проектировании многопроцессорных и многомашинных систем реального времени. Обработка информации и управления в реальном времени состоит в получении ВС информации от внешних объектов, вычислений и передач результатов на объекты не позже, чем это требуется для функционирования системы. Соответствующие расписания должны удовлетворять директивным ограничениям на сроки выполнения программ.

В рассматриваемых комплексах можно выделить группу программ управления вычислениями, подсистемы обмена информацией с местными и удаленными абонентами, отображения информации и диалога, обеспечения функционирования и решения функциональных программ. Работу таких программных комплексов могут эффективно реализовать мультиплексорные ВС, например

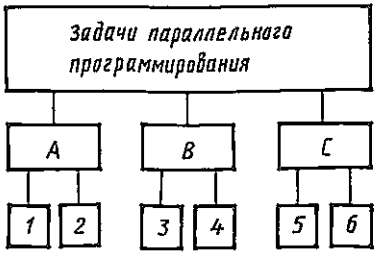


Рис. 7.8.
Классификация задач параллельного программирования.

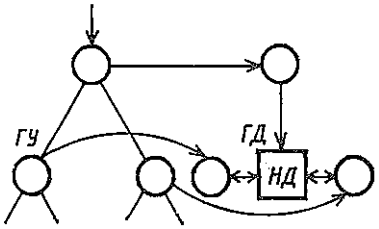


Рис. 7.9.
Графы управления и данных.

на базе микропроцессоров, в силу их производительности, надежности, живучести и готовности.

В цикле работ по проектированию программных систем реального времени целесообразно построить комплекс моделей программ, включая общую модель системы и подробные модели различных программ. Этот комплекс необходимо использовать для расчета характеристик и составления расписаний параллельных процессов. Модели строятся на основе методов и алгоритмов решения функциональных задач с учетом выбираемой или проектируемой ВС, а также на основе априорных данных о внешней среде. Выполнение расчета характеристик и построение вариантов расписаний позволяют определить производительность, тип и состав ВС, проанализировать варианты структуры и организации функционирования программ.

Схему работ по расчету характеристик ВПр можно представить следующим образом: создание программы расчета, составление моделей программ, ее корректирование, ввод, выполнение расчета на ЭВМ и получение результатов.

Модели программ. В таких моделях необходимо обеспечить отображение структур программ, вычислительных параметров (времени вычислений, объема памяти) и данных о путях выполнения программы. Последнее определяет отображение в моделях связей по управлению.

Графовые модели включают вершины, которым соответствуют компоненты программ, и дуги, отражающие связи между компонентами. Вычислительный процесс представляется в модели последовательностью переходов по дугам через вершины графов. При этом могут подсчитываться затраты машинного времени памяти, детализироваться логика.

В зависимости от циклов и ветвлений модели могут быть без упорядочения вершин, с частично упорядоченными вершинами (древовидные, сетевые), с ветвлениями, циклами, циклами и ветвлениями. По логике связей между вершинами модели могут быть без связей между вершинами, со связями на базе булевых функций И, ИЛИ, исключающее ИЛИ.

Рассмотрим представление структуры задачи моделью, включающей граф управления и граф данных (рис. 7.9). Граф управления (ГУ) используется для синхронизации выполнения программных компонентов и осуществления логических переходов. Граф данных (ГД) предназначен для интерпретации процесса обработки и передачи информации между компонентами. Вершинам графа управления ставятся в соответствие компоненты, а дугам — условия по пе-

редаче управления. Неоднозначное предшествование моделируется с помощью входной-выходной логики условий. Передача управления моделируется специальным знаком от входных дуг, при выполнении входной логики.

Кроме графовых моделей программ, различают модели на базе сетей Петри, марковские и полумарковские модели, имитационные модели, рассматриваемые в гл. 9.

Разработка параллелизма. В МВК параллелизм в той или иной форме применяется на уровне выполнения машинных команд. Поэтому при вычислениях возникает потребность в выражении параллелизма в алгоритме. На этапе разработки программы с момента постановки задачи можно выделить четыре стадии: 1) выбор подходящего алгоритма (А); 2) выражение алгоритма на языке (Я); 3) компиляция языка в объектную программу (О); 4) выполнение программы на машине (М). При идеальной ситуации число независимых операций, которые могут быть выполнены одновременно, не должно увеличиваться по мере разработки, т. е. А $\geqslant$ Я $\geqslant$ О $\geqslant$ М. К сожалению, большинство языков выражает параллелизм согласно ограничениям, которые заложены в объектной машине. С учетом огромного количества программ, разработанных для таких машин, возникает проблема построения компиляторов, выделяющих параллельные участки из последовательных программ. К таким компиляторам относятся векторизующие компиляторы, которые генерируют из последовательности операции параллельные машинные команды. Один из подходов заключается в выполнении оптимизации, основанной на представлении направленного графа в исходной программе, другой подход – в анализе и распараллеливании циклов DO в ФОРТРАНе (компилятор CFT GRAY-1).

Машинно-распараллеленные компиляторы строятся под определенную организацию МВК. Исходный язык выражает подлежащий рассмотрению параллелизм машины (язык для машины ILLIAC-IV и DAP-FORTRAN для ICL DAP).

Проблемно-распараллеленные компиляторы используют конструкции параллельности, заложенные в алгоритме. В частности, массивы могут рассматриваться как параллельные объекты данных, анализируются матричные и векторные операции и т. д.

Наконец, на верхнем уровне разработка параллелизма связана с выбором или синтезом параллельных алгоритмов. Эта проблема заключается в том, что если степень параллелизма в алгоритме соответствует степени параллелизма ВС, то опытный программист может написать высокопроизводительную программу, например в конвейерных ВС данные для операций обычно описываются как векторы, а выполняемые операции – как векторные команды.

Распознавание параллельных участков. Для идентификации задач и подзадач, которые могут выполняться параллельно, в язык программирования должны быть включены специальные операторы, которые снабжают ОС информацией, необходимой для управления параллельным режимом. Необходимо обеспечить: спецификацию начальных и конечных участков программ, выполняющихся параллельно; задания специальных условий для реализации или ограничения параллельного выполнения; задания условий для синхронизации параллельных процессов друг с другом и другими частями программы. Распознавание параллельных участков программ и определение их протяженности может быть осуществлено программистом или транслятором.

Большинство мультипроцессорных машин работают в целом эффективно, не используя возможности параллельного выполнения частей программы. Загрузка каждого процессора обеспечивается за счет наличия блока программ, каждая из которых выполняется последовательно. Программы из пула назначаются диспетчером. При этом дополнительные сложности возникают при распределении связанных программ, пути решения которых будут рассмотрены в § 7.4.

При использовании векторных, матричных или конвейерных ВС возникают специальные проблемы, решение которых зависит от количества матричных операций. В качестве операторов, которые могут служить для обеспечения параллельной обработки информации, применяются команды FORK и JOIN . Оператор ветвления L1 FORK L2, N указывает N независимых задач, готовых для параллельного выполнения, начиная с адреса L1, которые должны завершиться до оператора с меткой L2. Оператор объединения JOIN предназначен для указания частей программы, которые завершают параллельное выполнение. В языке ОККАМ имеется оператор PAR , сегменты после которого могут выполняться параллельно до оператора END .

В целом можно выделить три основных подхода при построении языков параллельного программирования: расширение существующих широко распространенных языков, построение специализированных с использованием устоявшихся языковых конструкций и построение новых языков. Примерами первого типа являются: расширение АЛГОЛа-60, расширение ФОРТРАНа, ПАСКАЛя и МОДУЛы, примерами второго типа – К-язык, ярусно-параллельные формы, PASOL , примерами третьего типа – СИМСКРИПТ, СИМУЛА-67 , АДА, ОККАМ.

7.4. ЗАДАЧИ РАСПАРАЛЛЕЛИВАНИЯ И МЕТОДЫ ДИСПЕТЧЕРИЗАЦИИ

Общие положения. С организацией параллельных вычислительных процессов связывают решение следующего класса задач: задачи точной оптимизации параллельной обработки в различной постановке; задачи построения диспетчеров ВС; задачи компоновки потоков заявок для оптимизации динамического распараллеливания; задачи выбора и оценки структур ВС, приспособленных к организации параллельных ВПр.

При построении структур ВС возникают задачи выбора числа вычислителей и закрепления за ними определенных задач с учетом информационной взаимосвязи и производительности вычислителей. На основании этого можно сформулировать две основные задачи параллельной обработки: 1) при заданных характеристиках вычислительных средств построить ВС минимальной стоимости, выполняющую заданный алгоритм за заданное время; 2) при заданных характеристиках ВС и реализуемого алгоритма найти план выполнения этого алгоритма за минимальное время. Решение указанных задач распараллеливания без учета информационных связей сводится к решению известных задач дискретного программирования. Наиболее трудоемки и наименее изучены задачи оптимального распараллеливания информационно-связанных компонентов.

Диспетчеризация может выполняться на уровне ОС (программные диспет-

черы) и ядра ОС (см. § 1.4). Рассмотрим виды и построение алгоритмов на уровне программ ОС.

Задачи диспетчеризации (составления расписания) могут быть классифицированы по принципам их построения, критериям оптимальности, выполнению этих критериев, методам выборов операторов и процессоров и назначению операторов на процессоры для выполнения обработки информации.

В зависимости от принципа построения расписаний можно выделить статические и динамические расписания. Первые строятся в период подготовки к решению задачи на ВС, а вторые – в процессе решения. Для многомашинных систем характерны статические расписания, для многопроцессорных – динамические.

В качестве основных критериев оптимальности расписаний для детерминированных моделей можно указать минимизацию времени выполнения программы, количества требуемых процессоров, среднего времени окончания выполнения задания, времени простоев процессоров, а также максимизацию загрузки процессоров.

По выполнению критериев оптимальности различают оптимальные расписания и приближенные. Первые используют метод динамического программирования для детерминированных и вероятностных моделей, программ и метод ветвей и границ. Полученные при этом комбинированные задачи имеют большую размерность и являются достаточно сложными. Это стимулирует разработку приближенных расписаний, среди которых можно выделить локально-оптимальные, обеспечивающие оптимальность на отдельных этапах, и эвристические. Во многих случаях затраты на построение оптимальных расписаний не окупают экономии при их реализации по сравнению с приближенными расписаниями.

При выборе операторов для назначения на процессоры используют структурные характеристики (связанность операторов, длину пути от данного до коконечного оператора) и временные (время исполнения, длительность критического пути).

При выборе процессоров в случае их однородности учитывается их минимальная занятость. В случае неоднородных выбирается процессор с наибольшей производительностью. При распределении операторов на процессор различают назначения: без прерываний и с прерываниями, незадержанное и задержанное.

Для оценки эффективности видов распараллеливания используются величины: $T_i(m)$ – время выполнения параллельной программы на m-процессорной ЭВМ при выполнении i-го расписания; $\tau(m)$ – минимальное время выполнения программы на m-процессорной ЭВМ (оптимальное расписание); $t(m)$ – время счета программы на однопроцессорной ЭВМ с производительностью m-продессорной. Эффективность m-процессорной ЭВМ по обеспечению распараллеливания процесса оценивается величиной $e(m) = t(m)/\tau(m)$. Эффективность i-го расписания выполнения параллельной программы на m-процессорной ЭВМ будет оцениваться выражением $f_i(m) = \tau(m)/T_i(m)$. Кроме того, при сравнении расписаний необходимо не только учитывать вышеприведенные показатели эффективности, но и затраты, требующиеся для реализации расписаний.

Рассмотрим модели распределения связанных работ (операторов, моду-

лей) с введениями оценки связанности, требуемым размером памяти и временными оценками распределяемых работ, если они заданы.

Постановка задачи. Пусть ВС объединяет n вычислителей. Исходное множество работ обозначим через $D = \{d_i\}, i = \overline{1, m}$. Зададим на D отношение частичного порядка R (в частном случае $R = 0$), состоящее в том, что $d_i R d_j$, если для работы d_j необходима выходная (полная или частичная) информация по работе d_i. Сопоставим отношению R ориентированный граф $\Gamma(D, W)$ с множеством вершин (работ) D и дуг W, причем $(\overrightarrow{d_i d_j}) \in W$, если $d_i R d_j$. Пусть граф Γ (D, W) не содержит контуров и петель. Относительно работ из D могут быть известными: требуемые размеры памяти M_i, $i = \overline{1, m}$; время выполнения работ на вычислителях t_{ij} (i – индекс работы, j – индекс вычислителя) или частичный порядок φ, заданный на подмножестве $D_\varphi \in D$, такой, что $d_i \varphi d_j (d_i, d_j \in D)$, если время выполнения работы d_i больше, чем время работы d_j.

Для определения отношения φ могут применяться как косвенные (предлагаемые), так и аналитические (вероятностные) оценки, основанные, например, на использовании дискретных марковских цепей. Косвенные оценки могут строиться исходя из требуемых размеров памяти M_i, т. е. можно положить $d_i \varphi d_j$, если размеры памяти (число операторов) работы d_i больше, чем размеры памяти работы d_j. Строго говоря, это не всегда верно, однако при отсутствии точных или приближенных значений t_{ij} подобное допущение имеет смысл.

В случае неоднородной ВС приходится иметь дело с вектором $\psi = (\varphi_{\alpha 1}, \varphi_{\alpha 2}, ..., \varphi_{\alpha p})$, где $\varphi_{\alpha f}$ – частичный порядок на $D_{\varphi f} \in D$, определенный для подмножества однотипных вычислителей a_f $(f = \overline{1, p})$.

Целью статической диспетчеризации является получение расписаний работ для каждого вычислителя с учетом критерия $T = \min(\max(\tau_k))$, где τ_k – время (с учетом простоев) работы вычислителя k.

При динамической диспетчеризации необходимо осуществлять распределение работ в процессе их выполнения на вычислителях, чтобы с учетом имеющейся (текущей) информации по возможности снизить непроизводительные затраты времени, связанные как с простоями вычислителей, так и с неоптимальностью произвольного распределения (например, при выполнении работ в порядке очередности).

Учет связности работ. Введем ряд определений. *Операцией разрешения связей* в графе (подграфе) Γ(D, W) назовем удаление вершин, принадлежащих некоторому подмножеству (например, подмножеству вершин, соответствующих работам, законченным к рассматриваемому моменту времени) и инцидентных им дуг. *Нулевым ярусом* $D^{(0)}$ графа Γ(D, W) (или его подграфа) назовем множество вершин (работ) с нулевой полустепенью захода. *Ярусом* $D^{(i)}$, $i = 1, 2, ...$ будем считать нулевой ярус подграфа $\Gamma_i(D_i, W_i)$, полученного в результате последовательного применения операции разрешения связей за счет нулевых ярусов образующихся подграфов графа Γ(D, W) ровно i раз.

Более простой подход может состоять в следующем. Множество вершин нулевого яруса $D^{(0)}$ разбивается на подмножества $D^{(0,0)}$, $D^{(0,1)}$, ..., $D^{(0,Q)}$, причем подмножество $D^{(0,q)}$ $(q = 0, Q)$ образует вершины нулевого яруса,

соединенные дугами с вершинами q-го яруса $D^{(q)}$. При распределении в первую очередь просматриваются вершины, принадлежащие подмножеству $D^{(0,1)}$, затем $D^{(0,2)}$, $D^{(0,3)}$ и т. д. В последнюю очередь распределяются вершины (работы) из $D^{(0,0)}$.

Отметим, что учет связанности планируемых работ позволяет повысить качество диспетчеризации в первую очередь за счет сокращения простоев вычислителей (случай, когда множество готовых к выполнению работ пусто).

Статическая диспетчеризация. При решении задачи статической диспетчеризации могут использоваться принципы поярусного распределения, позволяющие строить простые и легко реализуемые модели диспетчеризации, наиболее эффективные в условиях жестких ограничений на время планирования. Вершины (работы), входящие в ярус $D^{(i)}$ $(i = 0, 1, 2, ...)$, распределяются как независимые, что позволяет значительно упростить логику работы диспетчера. Для заданных значений t_{ij} (отношений φ_{ai}) субоптимальная процедура распределения готовых к выполнению работ может строиться согласно подходу, при котором в первую очередь распределяются наиболее трудоемкие работы и назначаются на вычислители, которые заканчивают их вычисление раньше других.

Указанная процедура может быть реализована следующим образом:

1. Полагаем $\tau_1 = \tau_2 = ... = \tau_n = 0$. Составляем из t_{ij} $(i = 1, m, j = 1, n)$ матрицу τ времени, в которой строки соответствуют номерам вычислителей, а столбцы – номерам работ рассматриваемого яруса, т. е. $\tau = [t_{ij}]$.

2. Если все столбцы помечены, процедура заканчивается. Иначе находим в каждом непомеченном столбце минимальный элемент и из этих минимальных элементов выбираем максимальный. Обозначим его $\tau(r, v)$. Столбец v помечаем. Работу, соответствующую этому столбцу, заносим в список работ вычислителя r. Полагаем $\tau_r = \tau_r + t_{vr}$.

3. Время t_{vr} добавляем ко всем элементам строки r, не входящим в помеченные столбцы. Переход к п. 2. В результате выполнения этого приближенного алгоритма вершины (работы) рассматриваемого яруса будут распределены с учетом оценки связанности по спискам соответствующих вычислителей. Для распределения всей исходной совокупности работ алгоритм выполняется для каждого яруса $D^{(i)}$ $(i = 0, 1, 2, ...)$. Качество планирования с использованием принципов поярусного распределения во многом определяется топологией связей и размерностью графа $\Gamma(D, W)$, поэтому трудно оценить потери времени, связанные с неоптимальностью распределения.

При другом варианте описанный выше алгоритм распределения несвязанных работ может выполняться на подмножествах $D^{(0,1)}$, $D^{(0,2)}$, ..., $D^{(0,Q)}$, $D^{(0,0)}$, что позволяет уменьшить затраты на диспетчеризацию, правда, при некотором снижении его качества.

Динамическая диспетчеризация. Использование описанных оценок позволяет упорядочить формирование входной очереди работ на обслуживание. Выделяются два списка $C_{г.р}$ и $C_{вх}$. Список готовых к выполнению работ $C_{г.р}$ формируется и динамически пополняется вершинами (работами) нулевых ярусов подграфов графа $\Gamma(D, W)$, которые получаются в результате операции разрешения связей на текущем подграфе $\Gamma_i(D_i, W_i)$ за счет работ, выполненных к рассматриваемому моменту. Работы в списке $C_{г.р}$ упорядочиваются по невозрастанию оценок связности. Входной список $C_{вх}$ пополняется работами

из $C_{г.р}$ в порядке их очередности. Максимальная длина (число элементов) списка $C_{вх}$ фиксирована и равна $2n$.

В случае однородной ВС список $C_{вх}$ упорядочивается отношением φ, если же это отношение не задано, то реализуется произвольное распределение работ, например по правилу FIFO.

Формирование списка $C_{вх}$ за счет работ из $C_{г.р}$ начинается при условии, если $C_{вх}$ пуст. При отсутствии головной ЭВМ (диспетчера) эту функцию реализует вычислитель, пытающийся получить код работы на выполнение из пустого списка $C_{вх}$. В случае, если таких вычислителей несколько, приоритет может отдаваться, например, вычислителю с большим номером или по другому правилу.

При выборке и назначении работ на выполнение из $C_{вх}$ осуществляется проверка наличия в системе свободной памяти требуемых размеров. Если на одной работе из $C_{вх}$ не может быть выделена требуемая память, то осуществляется просмотр списка $C_{г.р}$. При наличии в $C_{г.р}$ работы, которой может быть выделена требуемая память, эта работа помещается в начало $C_{вх}$. При этом если $C_{вх}$ содержал $2n$ элементов, то последний элемент возвращается в $C_{г.р}$.

Для неоднородной ВС логика формирования списков и выбор работ на обслуживание усложняются. Ниже рассматриваются два алгоритма динамического распределения работ для случая, когда заданы частичные порядки $\varphi_{af}(f=\overline{1,p})$ и когда никакой информации о временных оценках распределяемых работ нет. В последнем случае считается, что из любой пары вычислителей время реализации некоторой работы меньше у вычислителя, имеющего большую производительность (число операций в единицу времени). Тогда логика распределения в упрощенном виде реализуется следующим образом.

1. Первые n работ или любые n работ, для которых удовлетворяются требования по памяти, из $C_{вх}$ назначаются "один к одному" на n вычислителей. Если число работ в $C_{вх}$ меньше $n(n>l)$, то работы назначаются на наиболее быстродействующие вычислители из числа доступных.

2. Если ни один вычислитель не завершил выполнения текущей работы, то происходит ожидание, в противном случае обслуживаемая работа покидает систему, а незавершенные прерываются и их состояние запоминается. Разрешаются связи за счет законченных работ (т. е. формируется новый список $C_{г.р}$).

3. Если число незавершенных работ в $C_{вх}$ меньше n, то в $C_{вх}$ вводятся новые работы из $C_{г.р}$. Прерванные в п. 2 работы ставятся в конец списка $C_{вх}$, после чего осуществляется переход к п. 1.

Укажем, что п. 3 приведенного алгоритма может быть изменен таким образом, чтобы диспетчер стремился поддерживать постоянный размер списка $C_{вх}$, равный $2n$, за счет работ из $C_{г.р}$. Схематически работа алгоритма показана на рис. 7.10. Здесь I – текущая длина $C_{вх}$ $l<n$; II – смещение границы $C_{вх}$ при занесении работ из $C_{г.р}$; III – смещение границы $C_{вх}$ за счет постановки прерванных работ в конец очереди (при этом смещение II выполняется с учетом числа прерванных работ). При этом если длина $1>n$, то из $C_{г.р}$ работы не поступает, а смещение II выполняется за счет прерванных работ.

Рассмотренные методы обеспечивают возможность решения задач статической и динамической диспетчеризации для различных условий: типа ВС, наличия или отсутствия временных характеристик работ, связанности работ и др.

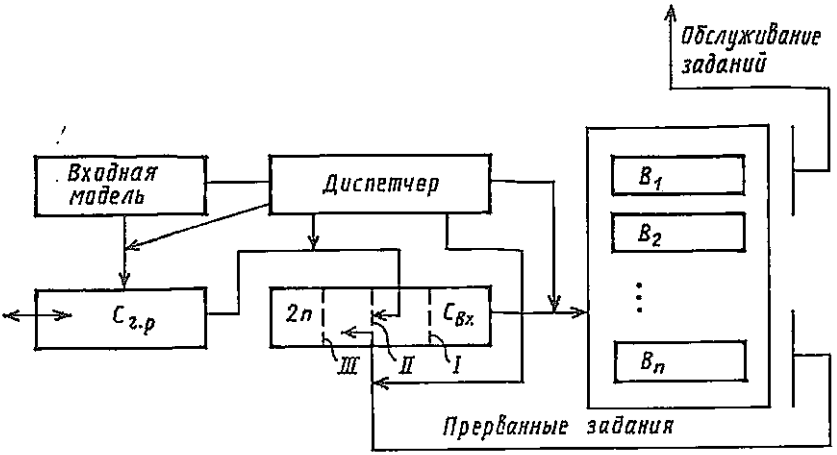


Рис. 7.10.
Схема динамического распределения связанных задач.

Сравнение вышеприведенных методов диспетчеризации с методами без учета связанности [4] показало преимущество первых от 7 до 12%.

Для программной реализации рассмотренных моделей статической и динамической диспетчеризации разработана следующая структура. Она включает модули: связи с ОС, разрешения связей, определения оценок связанности и сортировки, формирования расписаний, формирования входной очереди и выборки работ на обслуживание.

Для количественной оценки эффективности расписаний проводят эксперименты, основанные на программной имитации на ЭВМ процессов выполнения параллельных программ двумя или более процессорами ВС.

7.5. ОСОБЕННОСТИ ОС МНОГОМАШИННЫХ И МУЛЬТИПРОЦЕССОРНЫХ СИСТЕМ

Система ведущий–ведомый. Можно выделить три основных типа организации и способа функционирования мультипроцессорных ОС (хотя возможны их варианты с небольшими изменениями): ведущий–ведомый; раздельное выполнение заданий в каждом процессоре; однородная обработка во всех процессорах.

Выбор метода обработки в системе ведущий–ведомый может различаться, например функции управления выносятся на специальный супервизорный процессор. Супервизорные функции реализуются в специальном или универсальном процессоре. Супервизорные программы не обязательно должны быть реентерабельными, поскольку они выполняются одним процессором. В системе не возникают конфликты при доступе к системным таблицам. Возможны сбои в ведущем процессоре, что понижает отказоустойчивость системы. Система имеет малую гибкость из-за одного ведущего и нескольких ведомых процессоров (машин). К ведущему процессору предъявляются требования по

быстрому выполнению управляющих функций. Возможна перегрузка ведущего процессора и недогрузка ведомых. Время простоя, накапливаемое в ведомых процессорах, делает данный способ управления приемлемым для специальных применений, допускающих простои. Он эффективен также для неоднородных систем, состоящих из процессоров с существенно различающейся производительностью.

ОС с раздельным выполнением заданий. В этой системе основная память используется в режиме разделения, ОС может частично или даже полностью копироваться в каждом вычислителе. Возможна ситуация, когда каждый процессор работает автономно и выполняет супервизорные, исполнительные и вспомогательные функции, а также решает полностью задачу. Такая ОС характерна больше для многомашинной ВС и обладает следующими свойствами.

Каждый процессор обслуживает собственные потребности. Управляющие программы должны быть реентерабельными или дублироваться. Число конфликтов, связанных с блокировкой системных таблиц, невелико, поскольку каждый процессор имеет собственный набор таблиц.

Одиночные отказы не приводят к потере работоспособности, однако восстановление и запуск отказавшего процессора трудоемки. Практически каждый процессор имеет свои собственные устройства ввода-вывода, файлы, через которые возможно взаимодействие. Такая система может иметь невысокую эффективность из-за неравномерной загрузки процессоров.

ОС с однородной обработкой. В такой системе каждый процессор может выполнять в равной мере супервизорные функции. Исполнительные функции переходят из одного процессора в другой. Для ОС характерны следующие свойства. Функции ведущего передаются с одного процессора на другой. Любая функция может выполняться любыми процессорами. Управление непрерывно перераспределяется между процессорами: один из процессоров является исполнительным, в каждый момент времени только один процессор выполняет планирование; для процессоров может быть установлен приоритет с целью разрешения конфликтных ситуаций и ранжировки управляющих функций. Программные модули являются реентерабельными для выполнения различными процессорами. Важной проблемой является доступ к системным таблицам и наборам данных, для чего требуется механизм синхронизации. Система обладает несколькими преимуществами: сохраняется работоспособность при выходе из строя отдельных устройств, эффективнее осуществляется загрузка ВС, реализуется резервирование системы для повышения надежности. Эта система – основа ОС распределенной обработки.

· **Требования к функциональным характеристикам.** Кроме функций однопроцессорной ОС, в многопроцессорной системе реализуется ряд дополнительных задач. При работе с памятью предусматривается ее разбиение на блоки, сегменты, страницы и реализация переадресации страниц (в процессоре или памяти). Планировщик должен определять взаимосвязи между различными задачами, а диспетчер работать как с несвязанными, так и со связанными задачами. При этом должна быть равномерной загрузка процессора в каждый период времени.

При выполнении одного задания несколькими процессорами необходима взаимосвязь друг с другом. Эти связи обеспечиваются комбинацией аппаратных и программных средств (связь посредством прерываний, через почтовые

ящики и т. д.). Требуются дополнительно аппаратно-программные средства защиты системных таблиц и НД от неправомерного доступа. Необходимо создать балансировку загрузки системы ввода-вывода (приоритетный принцип для формирования очередей запросов и т. д.). Особенно важным является обеспечение аппаратной реконфигурации системы для повышения ее надежности и производительности. Наконец, проблема тупиков при работе со всеми видами ресурсов в мультипроцессорной ОС должна решаться более надежными средствами.

К основным функциям многопроцессорной ОС на уровне ядра могут быть отнесены следующие: работа с параллельными процессами (создание, окончание, изменение приоритета и удаление процесса), работа с очередями (создание очереди, чтение, запись очереди), задержка, диспетчеризация, работа с консолями (назначение, установка, подключение и отключение), вызов системной процедуры, работа с дескрипторами процессов.

Особенности ОС в распределенных системах. При создании вычислительных систем принцип параллельной обработки реализуется как на уровне структуры, так и на уровне системного программного обеспечения. Для сильносвязанных параллельных систем можно выделить ограничения, влияющие на построение управляющих программ: непосредственное разделение ресурса приводит к конфликтам доступа и к задержке получения ресурса; языки программирования для сильносвязанных систем разработаны недостаточно; любые неэффективности ОС увеличиваются. Разработка оптимальных планов загрузки процессоров — задача очень сложная. В слабосвязанной многомашинной системе эти ограничения ослабляются. Однако при выполнении обменов необходима реализация одним процессором чтения, а вторым — записи. Кроме того, один процессор не может заставить второй что-либо делать. Таким образом, будучи слабосвязанными физически, процессоры благодаря ОС сильно связаны логически.

Рассмотрим особенности построения ОС с децентрализованным управлением и распределенными функциями. Распределенное управление характеризуется размещением физических компонентов в различных узлах, при этом имеется много точек управления. В качестве модели для разработки распределенной ОС служит модель для связи открытых систем. Эта модель имеет семиуровневую структуру (табл. 7.1), однако из нее можно использовать два существенных момента: описание связи между узлами и рекомендации по протоколам более высокого уровня. Поэтому модель распределенной системы будет включать следующие уровни: физический (передача сигналов), канальный (передача данных), сетевой (управление передачей между узлами), транспортный (управление сеансовой передачей, сетевыми соединениями), представительский (коды передаваемых данных), сетевой ОС и пользователя. Сетевая ОС делится на две части: 1) локальную ОС, отвечающую за управление ресурсами и обслуживание пользователей в узле; 2) распределенную ОС, отвечающую за взаимодействие данного узла с остальными. В соответствии с уровнями модели распределенной системы можно поставить ее компоненты. На прикладном уровне расположены пользователи и ресурсы, локальная и сетевая ОС. На представительском, сетевом, транспортном уровнях находится обработчик сообщений. Средство доставки сообщений работает на сетевом, канальном и физическом уровнях.

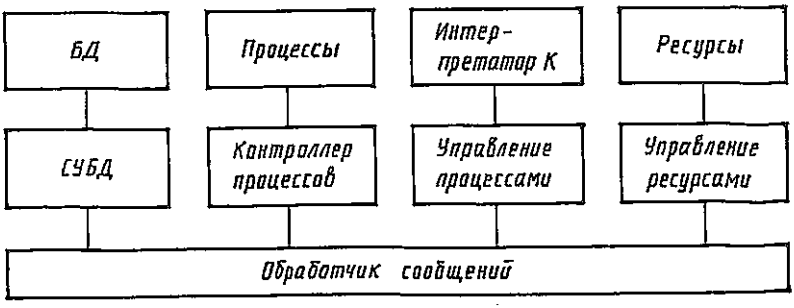


Рис. 7.11.
Упрощенная структура распределенной ОС.

Таблица 7.1

Протоколы						
Верхний уровень			Нижний уровень			
7	6	5	4	3	2	1
Язык пользователя	Язык связи сети	Язык сеансов связи	Транспортный	Сетевой	Канальный	Физический
Пользовательский	Представительский	Сеансовый	Заголовок передачи	Заголовок пакета	Заголовок блока	Начало, конец
			Фрагменты	Пакеты	Блоки	Биты

Окончательно структура распределенной ОС может быть представлена в виде, изображенном на рис. 7.11. На логическом уровне находятся база данных, процессы, ресурсы, интерпретатор команд. На физическом уровне имеются СУБД, контроллер процесса, управление процессами и ресурсами.

Управление в распределенной системе. Рассмотрим вопросы управления в такой системе с трех точек зрения: влияние динамики системы на управление; информация, которую поддерживает элемент управления; принципы проектирования управления.

В распределенной системе рабочие запросы могут поступать от пользователя либо быть образованы от активных процессов и поступать в любой процессор. В принципе запросы могут потребовать любой процессор, поэтому необходима поддержка разнообразных запросов, поступающих в различные устройства от любых источников.

При работе с ресурсами запрос на них может возникнуть в любом узле. Поскольку ресурсы могут отказать, необходимо управление ими в условиях непредсказуемости их доступности. Наконец, если запросы прямо или косвенно определяют множество взаимодействующих процессов, каждый из этих процессов может работать с файлами, содержащими дополнительные рабочие

запросы. Необходимо управлять этими процессами так, чтобы обеспечить в системе максимально возможный параллелизм в работе. Для выполнения своих функций система управления использует определенные структуры данных, имеющие особенности. В силу характера связей в системе рассматриваемая информация является устаревшей. Процессор может принимать решения на запрос самостоятельно.

Поступивший прикладной процесс представляет запрос, для которого система управления выполняет три задачи: сбор информации, распределение работ и назначение ресурсов, исполнение заданий. *Сбор информации* состоит из определения потребности задания и выявления доступных ресурсов для его реализации. Если *распределение работ* выполнить сразу не удается, то возможны три варианта: 1) собрать больше информации, но по-новому распределить работу; 2) собрать больше информации о ресурсах и выполнить работу за большее время или с увеличением ресурсов; 3) информировать пользователя о невыполнении запроса. Если различные части запроса реализуются на различных процессорах, то необходимо исправлять некоторые отказы. Это возлагается на средства восстановления, которые информированы о размещении работы в узлах и имеют возможности связи с новыми узлами для окончания части работы, выполнявшейся на отказавших процессорах.

Граф заданий строится при поступлении запроса, хранится целиком или по частям в узлах и создается до начала исполнения или во время исполнения. Информация о доступности ресурсов собирается одним узлом либо несколькими по видам ресурсов. Распределение ресурсов решается предварительным резервированием. При запуске процесса можно сделать один узел ответственным за весь граф задания или несколько узлов. При отказе узла можно повторять запросы или обратиться с запросом к другому узлу. При управлении процессами их взаимодействие осуществляется через порты, а обработка запросов выполняется либо в одном, либо в нескольких узлах. После выполнения работ производится очистка памяти и информирование всей системы о завершении работы.

Выводы

1. Многомашинные и многопроцессорные ВС (МВС) могут строиться в зависимости от типа потока команд (одиночный и множественный); типа потока данных (одиночный и множественный); числа одновременно обрабатываемых разрядов (словные и одноразрядные); степени связанности блоков (сильносвязанные и слабосвязанные); степени однородности, средств комплексирования (процессоров, ОП, каналов, внешних устройств). Связь в МВС может организовываться через общие или раздельные шины и коммутаторы. Основные типы МВС: векторные, матричные, ассоциативные, с перестраиваемой структурой.

2. Параллельные ВС на базе ЕС ЭВМ комплексируются на уровне процессоров (прямое управление), каналов, внешних устройств (многомашинные ВС) и общей памяти (многопроцессорные ВС, на примере ЕС 1065, ЕС 1066).

МВК "Эльбрус", включающий несколько центральных процессоров, ввода-вывода, связных, блоки памяти, периферийные устройства, относится к типу МКМД. Вычислительный процесс на логическом уровне организуется путем распределения процессов, адресуемых через стек, на свободные процессоры.

3. Для параллельных вычислений программы могут представляться графовыми или матричными моделями. Распараллеливание осуществляется на уровне исходного алгорит-

ма, языка программирования, компилятора и машинных команд. Это может реализовываться соответственно программистом; разработкой новых или модификацией известных языков; векторизующими, машинно- и проблемно-ориентированными компиляторами; аппаратно-программными (ОС) средствами МВС.

4. Диспетчеризация на логическом уровне реализуется средствами ОС (программно) и на уровне ядра ОС (микропрограммно или аппаратно). Программные диспетчеры могут быть статические и динамические, для связанных и несвязанных задач, точные и приближенные для однородной и неоднородной ВС.

5. Особенностью ОС МВС является расширение ее функций как на уровне ядра, так и на уровне работы программиста.

Рекомендуемая литература: [3, 4, 14, 15, 23].

Вопросы для контроля

1. Поясните разницу между Эрланской классификацией ВС и по Флину.
2. В чем особенность организации вычислительных процессов в МВС типа: а) МКОД; б) ОКМД; в) МКМД?
3. Поясните разницу в организации ВС на базе ЕС и "Эльбрус".
4. Каким образом распознаются параллельные участки в программах?
5. В чем особенность графовых моделей программ?
6. Поясните сущность статической диспетчеризации.
7. Поясните идею динамической диспетчеризации.
8. Объясните назначение уровней в распределенной ОС.

8. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ МИНИ- И МИКРОЭВМ

8.1. СОСТАВ И ФУНКЦИИ ПО МИНИ- И МИКРОЭВМ

Программное обеспечение СМ ЭВМ. В настоящее время в нашей стране в основном используются два класса мини- и микроЭВМ в зависимости от системы команд. К первому классу относятся мини- и микроЭВМ с системой команд СМ-4: "Электроника-60", СМ-1420, ДВК-2, ДВК-3, ко второму – ЭВМ с интеловской системой команд: СМ-1800, ЕС 1840 и т. д.

ПО строится как совокупность различных по своим возможностям ОС и пакетов прикладных программ. Для СМ ЭВМ характерно использование десяти ОС, нескольких десятков пакетов, а также тест-мониторной системы.

Конкретная ориентация систем определяется прикладными программными средствами пользователя. ОС СМ ЭВМ по своему назначению делятся на три класса: общего назначения, реального времени и разделения времени. ОС общего назначения включают перфоленточную систему (ПЛОС СМ), диалоговую систему программирования (ДС СМ) и дисковую операционную систему (ДОС СМ). Они обеспечивают организацию вычислительных процессов, подготовку, отладку и выполнение программ пользователей, управление вводом-выводом. ОС реального времени включают четыре системы: перфоленточную ОС реального времени (ПЛОС РВ), фоново-оперативную базовую ОС реального времени (ФОБОС), дисковую ОС реального времени (ДОС РВ) и ОС реального времени (ОС РВ). В состав ОС этого класса входят мониторные средства, обеспечивающие одновременное выполнение нескольких задач реального времени. ОС разделения времени представлены дисковой диалоговой многотерминальной системой (ДИАМС) и дисковой ОС разделения временных ресурсов (ДОС РВР). Эти системы обеспечивают для многих пользователей, в том числе удаленных, разделение ресурсов мини-ЭВМ. Для таких ОС характерно: разделение устройств (процессоров, памяти, ввода-вывода); разделение файлов (в том числе для удаленных пользователей); защита доступа (данные и программы защищаются от несанкционированного доступа); наличие диалоговых языков программирования; расширенный состав программных средств телеобработки данных.

Существенными компонентами ОС СМ ЭВМ являются средства автоматизации программирования, включающие трансляцию (интерпретацию с входных языков), редактирование связей и отладку программ. Среди языков следует отметить возможность диалоговой работы (ДС СМ, ДИАМС, БЕЙСИК), что ведет за собой применение интерпретаторов. При этом для ОС интерпретатор является и мониторной системой.

Наиболее распространенными языками программирования являются макроассемблер, ФОРТРАН, в последнее время ПАСКАЛЬ и БЕЙСИК. Трансляторы, кроме перевода программ в машинные коды и синтаксического контроля, обеспечивают выдачу объектного модуля на перфоленту.

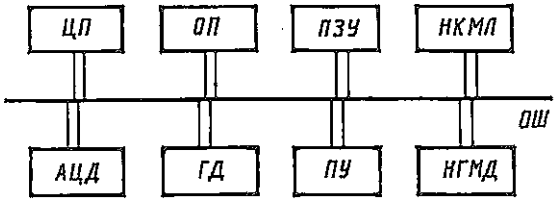


Рис. 8.1.
Упрощенная структура персональной ЭВМ.

Многопользовательскими являются системы ДОС РВ, ОС РВ, ДИАМС, ДОС РВР. Наличие средств телеобработки данных определяет возможность доступа удаленных пользователей к ресурсам.

В состав программного обеспечения СМ ЭВМ входят процедурно-ориентированные и проблемно-ориентированные пакеты. Процедурно-ориентированные пакеты включают: реализации функций технологии обработки данных; систему телеобработки данных, управление базами данных в системе ДИАМС; пакет для ведения банков данных на иерархических комплексах. Пакеты, реализующие логико-математическую обработку данных, содержат пакет программ численного анализа, обработки данных методами математической статистики, пакет программ методов оптимизации.

В состав проблемно-ориентированных пакетов входят пакеты: имитационного моделирования непрерывных и дискретных процессов; экономических применений; обработки данных в системах оптимизации лабораторных экспериментов; научно-технических расчетов.

Программные средства микроЭВМ. В связи с ускоренным развитием персональных микроЭВМ с начала 80-х годов быстрыми темпами развивается их аппаратное и программное обеспечение. Структура персональных ЭВМ включает центральный процессор (ЦП), оперативную и постоянную память объемом от 64 К байт до 1 М байт, алфавитно-цифровой (АЦД) или графический (ГД) дисплей, накопители на гибких магнитных дисках (НГМД) (207, 133 и 76 мм) (рис. 8.1). К процессору могут подключаться: печатающее устройство (ПУ), накопитель на кассетной магнитной ленте (НКМЛ), контроллер локальной сети, графопостроитель и другие внешние устройства. Центральный процессор строится на базе 8-, 16- или 32-разрядных микропроцессоров. Подключение периферийных устройств производится через контроллеры интерфейсов или в мощных ЭВМ через процессоры ввода-вывода к общей шине (ОШ).

Общее программное обеспечение для персональных ЭВМ условно можно разделить на три уровня: операционные системы, системы программирования и прикладные программы. ОС персональных ЭВМ выполняют следующие функции: управление процессором, памятью, периферийными устройствами; распределение ресурсов между задачами и поддержка взаимосвязи задач. В зависимости от количества задач и пользователей различают однопользовательские однозначные ОС (CP/M, MS-DOS 1.0), с фоновой печатью (MS-DOS 2.0); однопользовательские многозадачные (CP/M-86); многопользовательские многозадачные (MP/M-86, UNIX).

Структурно ОС включают базовую систему ввода-вывода (BIOS) для управления печатью, клавиатурой, дисплеем, НКМЛ; систему управления памятью на НГМД, ядро (ДОС), файловую систему, систему общения с пользователем – командный язык. Основная роль ОС заключается в эффективном обеспечении трех интерфейсов: с аппаратурой, системными программами и пользователем. Системы программирования микроЭВМ поддерживают программирование на ведущих языках: БЕЙСИК, ПАСКАЛЬ, ФОРТРАН, КОБОЛ, язык С. БЕЙСИК привлекает пользователей своей простотой и диалоговым режимом. ПАСКАЛЬ позволяет строить структурированные программы, имеет средства для описания данных различных типов, в том числе вводимых пользователем. ФОРТРАН предназначен для решения научных и инженерных расчетов, КОБОЛ – для решения экономических задач. Для микроЭВМ используется сокращенная версия МИБОЛ. Язык С является языком ОС UNIX. Кроме того, для программирования используются языки PL/M, АПЛ, а в последнее время АДА и ОККАМ. В систему программирования микроЭВМ также входят редакторы текста, компоновщики (редакторы) связей, загрузчики, отладчики, программы сортировки и управления файлами.

Для персональных ЭВМ разработан ряд пакетов: для обработки текстов (Wordstar, Lisa Write); системы машинной графики (GSS, GSX, Lisa Drow); системы, поддерживающие коммутацию в локальных сетях (Ethernet, Applenet); системы управления базами данных (dBase-II, Lisalist); системы табличного ввода и обработки данных (Lisacalc, Visicalc); системы деловой графики (Lisagraph), Lotus 1-2–3.

8.2. ОПЕРАЦИОННЫЕ СИСТЕМЫ МИНИ- И МИКРОЭВМ

ОС СМ ЭВМ. *Диалоговая система (ДС СМ)* представляет собой диалоговый язык и программу-интерпретатор с этого языка. Она предназначена для подготовки, отладки и выполнения программ пользователя, написанных на входном диалоговом языке.

ДС СМ позволяет работать пользователю в диалоговом и пакетном режимах, обеспечивает обработку чисел с фиксированной и плавающей запятой, выполнение стандартных функций, предоставляет сервисные средства для подготовки и отладки программ. Система выполняет следующие функции: подготовку программ на входном языке ДС СМ; корректировку программ в режиме диалогового редактирования текста; трассировку – управление распечаткой текста программ; загрузку в память с перфоленты программ пользователя и вывод программ на перфоленту или печать.

ДС представляется на перфоленточном носителе. Для функционирования ДС необходима память не менее 16 К байт, ввод-вывод с перфоленты, дисплей на базе Видеотон-340. Области применения ДС: автоматизация научно-технического эксперимента, научно-технические и экономические расчеты, обучение.

Перфоленточная ОС (ПЛОС) СМ представляет собой комплекс программ, предназначенных для подготовки, отладки и выполнения пользовательских программ на языке ассемблера в однопрограммном режиме, а также в многопрограммном для системы ПЛОС РВ. Областью применения ПЛОС являются простые системы управления, научно-технические расчеты, автоматизация на-

учного эксперимента. В системе все программы находятся на перфоленте. Ввод-вывод программ и данных выполняется через соответствующие периферийные устройства. Диалог пользователя с системными программами осуществляется с помощью системы команд, вводимых с клавиатуры терминала. Минимальный объем требуемой для ПЛОС памяти составляет 16 К байт.

Дисковая ОС (ДОС СМ) является системой общего назначения, предназначенной для разработки и отладки программ в диалоговом режиме. ДОС обеспечивает следующие возможности: размещение на дисках файлов системы и пользователя с применением многоуровневых каталогов; копирование, распечатку и защиту файлов; написание, трансляцию, компоновку и отладку программ на языках ФОРТРАН-IV или макроассемблера; редактирование символьных файлов и создание библиотеки объектных и загрузочных модулей. ДОС содержит набор управляющих программ, построена по модульному принципу, включает набор обрабатывающих программ для работы с файлами и ведения библиотек. Минимальный объем памяти, требуемый для функционирования системы, составляет 32 К байт.

Перфоленточная ОС реального времени (ПЛОС РВ) предназначена для решения широкого класса задач, возникающих в системах автоматизации научных экспериментов и системах управления технологическим оборудованием. Система обеспечивает мультипрограммный режим на приоритетной основе с использованием средств планирования задач, а также связь системы управления с оператором. ПЛОС полностью размещается в основной памяти в процессе функционирования и требует объема памяти 16 К байт. Система включает средства, реализующие операции ввода-вывода, обеспечивающие независимость задачи от периферийных устройств.

ОС представляет следующие возможности планирования задач: одновременное выполнение до 127 задач реального времени; выполнение одной фоновой задачи; выполнение задач на четырех программных приоритетных уровнях; инициирование задач по запросу оператора, программному запросу, прерыванию и запросу от таймера. Продолжительность пребывания задач на каждом приоритетном уровне определяется таймером, который понижает приоритет задачи, если время ее выполнения превышает заданный для этого уровня интервал.

Кроме управления в реальном времени, ПЛОС обеспечивает средства разработки ПО с использованием редактора текста, перемещаемого ассемблера и компоновщика, которые оформляются как фоновые задачи.

Фоново-оперативная базовая ОС РВ (ФОБОС) предназначена для создания конкретных версий ОС проблемно-ориентированных комплексов на базе СМ-3, СМ-4, ДВК. Областями ее применения являются автоматизация лабораторных экспериментов, управление испытаниями, решение вычислительных задач. ОС характеризуется минимальным по сравнению с другими системами реального времени периодом ответа за внешнее воздействие. Резидентная часть супервизора ФОБОС занимает 8 К байт памяти, предоставляя остальную часть памяти для ПП. Общая необходимая память для работы системы составляет 32–56 К байт.

Система обеспечивает три модификации операций ввода-вывода: синхронный – без возврата управления запрашиваемой программе; асинхронный – с немедленным возвратом управления после постановки запроса в очередь;

с событием, при котором запрос устанавливается в очередь, после его обслуживания управление возвращается вызывающей программе.

ФОБОС поддерживает систему программирования на языках макроассемблера, ФОРТРАН-IV, ДИАСП при работе в диалоговом режиме. Имеется монитор, который объединяет несколько программ в пакет и производит его обработку. Пользователи имеют возможность выполнять сравнение текстов, преобразование файлов, изменение кодов и объектных модулей, печать в различных форматах.

Дисковая ОС реального времени (ДОС РВ) является мультипрограммной системой с фиксированным уровнем приоритетов. Она предназначена для создания систем автоматизации научных исследований и проектно-конструкторских работ, является резидентной ОС, занимает до 16 К байт памяти. Поддерживает систему программирования на ФОРТРАНе-IV , расширенного средствами для работы в реальном времени. При этом трансляция и отладка программ производятся в ДОС СМ, поскольку имеется совместимость на уровне программных запросов ввода-вывода, форматов загрузочных и объектных кодов файловой структуры на дисках.

Для связи оператора с системой и задачами имеется набор команд пользователя. ДОС РВ обеспечивает следующие возможности: выполнение на приоритетной основе множества задач реального времени; выполнение одной фоновой задачи; реализацию задач на четырех программных приоритетных уровнях. Продолжительность задачи на каждом приоритетном уровне задается при генерации. На низшем уровне время выполнения не ограничено.

ОС реального времени (ОС РВ) представляет собой дисковую систему, обеспечивающую решение задач управления. Версии системы генерируются от небольших систем для лабораторных исследований до больших многопользовательских систем обработки и управления.

Параллельное выполнение многих задач в режиме реального времени обеспечивается на основе приоритетной диспетчеризации, разбиении памяти на разделы, временной выгрузке задач на диск. Система обеспечивает обслуживание многих терминалов, причем любой из них можно использовать в качестве командного. В системе могут выполняться задачи реального времени и фоновые, разработанные с применением языков макроассемблера и ФОРТРАНа-IV.

Основные компоненты ОС РВ обеспечивают: мультипрограммирование, приоритетную диспетчеризацию с квантованием времени; выходы из синхронных и асинхронных прерываний; динамическое распределение памяти с ее разбиением на разделы автоматическим уплотнением; взаимодействие задач и реактивность их при запуске. При работе с файловой системой имеется возможность использовать различные диски, преобразовывать файлы в форматы ДОС, ФОБОС и наоборот, организовывать работу с резидентными библиотеками модулей. Для системы ввода-вывода характерным является динамическая реконфигурация устройств, многотерминальная работа, динамическая загрузка и выгрузка задач на диск. Версия ОС РВ с диспетчером памяти требует 56 К байт памяти, без него от 32 до 56 К байт.

Дисковая ОС разделения временных ресурсов (ДОС РВР) служит для подготовки, отладки и выполнения программ. Ее области применения — управление научным экспериментом, системы оперативного управления, обучение, решение научных задач. Языком программирования является БЕЙСИК с набо-

ром функций и матричных операций. Одновременно может работать до 24 пользователей. Система осуществляет многозадачную работу в диалоговом режиме с разделением времени и обеспечивает защиту памяти и работу с удаленными терминалами через телефонные или телеграфные линии связи. Минимальный объем памяти системы 80 К байт.

ОС микроЭВМ. Для 8-разрядных микроЭВМ чаще используется одна из двух типов диалоговых ОС CP/M, MP/M. Система CP/M отличается простотой и большой гибкостью, является стандартом для 8-разрядных микропроцессоров. Разрабатываемые ОС обеспечивают среду CP/M. ОС MP/M – расширенный вариант CP/M для нескольких пользователей. Она более эффективна для 16-разрядных микроЭВМ, поскольку из-за относительной сложности ее использование на 8-разрядных МП приводит к снижению пользовательских характеристик (время реакции).

Для 16-разрядных микроЭВМ используются следующие типы диалоговых ОС: CP/M-86, MP/M-86, MS-DOS и UNIX . ОС CP/M-86 является развитием ОС CP/M и более эффективна для мультипроцессорной микроЭВМ на базе МП I8086, I8088 и отечественного К1810ВМ86.

ОС MP/M-86 является вариантом системы MP/M для многопрограммного и многозадачного режимов и пригодна для работы с несколькими пользователями. Она обеспечивает максимальное использование на 16-разрядных микроЭВМ (может использоваться на 8-разрядных) благодаря широким возможностям адресации и быстроте обработки запросов пользователей. Эта система открывает ряд новых возможностей по управлению ресурсами, сетевыми средствами, файловыми системами и т. д. В целом MP/M-86 построена по модульному принципу и ориентирована на управление процессами, а не задачами.

ОС MS-DOS разработана для персональных компьютеров фирмы IBM и может работать на отечественных персональных ЭВМ. По своим функциям она аналогична ОС CP/M, но пользователю предоставляются более широкие возможности по работе с графическими данными.

ОС UNIX предназначена в основном для 16-разрядных (и 32-разрядных) микроЭВМ, хотя первоначально она была разработана для мини-ЭВМ PDP-11. Ядро системы написано на языке С, поэтому ОС легко модифицируется и переносится с одной машины на другую.

8.3. ОРГАНИЗАЦИЯ ОС CP/M И MP/M

Состав ОС. Типичным представителем однопрограммной ОС для 8-разрядных микроЭВМ является CP/M (для 16-разрядных CP/M-86). Конструктивно ОС состоит из базовой ДОС (BDOS), базовой системы ввода-вывода (BIOS) и интерпретатора приказов ССР. CP/M размещается в верхней части основной памяти, а область пользователя (транзитная область программы TPA) – в нижней. В начальной области памяти располагаются векторы системных вызовов – команды условного перехода на соответствующие модули (рис. 8.2).

Базовая ДОС. Включает ядро системы и управление дисковыми файлами. Ядро обеспечивает доступ ко всем устройствам системы. Независимо от выполнения прикладная программа обращается к ядру для получения обслуживания. Для простоты интерфейса с ядром последнее содержит программу системных ресурсов, которая располагается в начале BDOS. Этой программе пере-

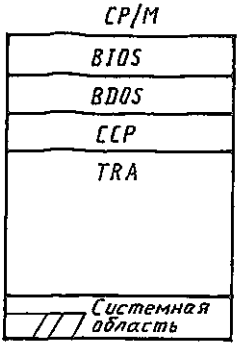


Рис. 8.2.
Организация основной памяти ОС CP/M.

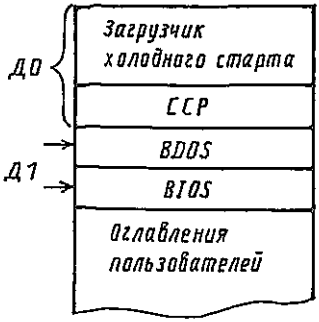


Рис. 8.3.
Распределение внешней памяти ОС CP/M.

дается номер запрашиваемой функции через один из общих регистров. По номеру осуществляется переход на требуемый и начальный адреса программы, реализующей функции системы. К этим функциям относятся: 0 — системный сброс, 1 — ввод с консоли, 2 — вывод на консоль, 3 — ввод со считывателя, 4 — вывод на перфоратор и т. д., 14 — выбор диска, 15 — открытие файла, 16 — закрытие файла.

Доступ к файлу диска осуществляется через блок управления файлом (FCB), который содержит имя файла, экстенты (расширение описания файла), карту распределения секторов и т. д. (32 байта). После обращения к файлу происходит вызов BDOS, которая загружает в FCB всю информацию, необходимую для доступа к файлу. Дисковое пространство распределяется блоками по 1 К байт. Файл может содержать до 64 К секторов, по 128 байт каждый. Распределение диска в CP/M показано на рис. 8.3. Нулевая (Д0) и первая (Д1) дорожки содержат загрузчик и систему CP/M. Затем следует оглавление на 64 элемента и файлы пользователей.

Секция базового ввода-вывода управляет системной консолью, печатающими устройствами и ленточным накопителем. Ядро, приняв запрос на обслуживание, находит адрес функции и передает управление подпрограмме ввода-вывода. Эти подпрограммы включают управление байтовым и дисковым вводами-выводами. При работе с диском выполняются следующие функции: выбор диска, установка дорожки, установка сектора, считывание сектора, запись сектора и т. д.

Интерпретатор приказов предназначен для управления системой путем представления пользователю набора функций. Последние в зависимости от реализации бывают резидентными (находятся в памяти) и транзитными (располагаются в командном файле на диске COM).

К резидентным относятся приказы для работы с файлами: DIR — отображает список файлов, ERA — удаляет файл, TYPE — отображает файл, SAVE — сохраняет созданный файл на диске, REN — переименовывает файл. К транзитным относятся файлы трансляторов — ASM (АССЕМБЛЕРа), FORT (ФОРТРАНа), PLM (ПЛ/М); редактора текста — ED, загрузчика — LOAD и т.д. Тран-

зитные приказы при их вызове загружаются в память. Они могут быть и прикладными программами, написанными пользователем. Когда пользователь вводит приказ, интерпретатор просматривает внутреннюю резидентную таблицу для встроенных функций. При отсутствии в таблице требуемой функции интерпретатор начинает поиск в оглавлении диска имен файлов. Обнаружив файл с именем, содержащимся в приказе, интерпретатор загружает этот файл в область программ и выполняет его.

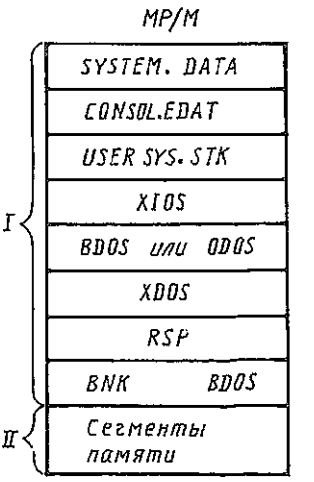


Рис. 8.4.
Внутренняя организация ОС MP/M.

Многопользовательская ОС. Система MP/M является расширением CP/M и дает возможность пользователям перейти к мультипрограммной обработке. Основу системы составляет многозадачное ядро, обеспечивающее реализацию разнообразных функций. К основным функциям ядра относятся: планирование процессов, управление очередью, памятью, флажками и системным эталоном времени. В системе предусмотрено приоритетное выполнение задач от 0 (наивысший) до 255. Задача с наивысшим приоритетом получает процессор на время, пока не возникнет сигнал прерывания или она не обратится к BIOS. В последнем случае происходит выполнение функции планирования, отмечается задача с наивысшим приоритетом, ей передается управление. Задачи с равным приоритетом получают равный квант времени.

Ядро также управляет памятью. Для этого адресное пространство делится на сегменты, в каждом из которых может находиться прикладная либо системная программа. Для управления ресурсами осуществляются построение и обработка очередей (типа FIFO), предназначенные для организации взаимодействия и синхронизации задач. Применение флажков в системе позволяет реализовать систему логических прерываний, которые могут отличаться от физических прерываний. Установка флажка соответствует событию в среде реального времени и вызывает выполнение функции ядра. Управление временем включает поддержание времени дня, планирование работ в реальном времени и осуществление задержки задач.

В структурном отношении MP/M аналогична CP/M, но содержит больше модулей и вариантов. Внутренняя организация системы показана на рис. 8.4. В верхней части памяти располагается область системных данных (*I*) и пользователя (*II*). SYSTEM.DATA определяет конфигурацию MP/M по заданию пользователя. Генерация варианта ОС производится с помощью специальной программы GENSYS. Затем следует область данных консоли, которая содержит информацию для управления консолями, подключенными к системе.

Системная область пользователя USERSYS.STK необходима только для работы с командным файлом COM. Если выполняются странично-перемести-тельные файлы, то последняя область отсутствует. Модуль XIOS представляет собой расширенную версию базового ввода-вывода CP/M и включает BIOS

CP/M. Базовая система BDOS, как и в CP/M, содержит функцию управления файлами. В систему с коммутацией банков вместо BDOS входит ODOS . Базовая DOS управляет также консолями. Расширенная версия базовой DOS(XDOS) работает с учетом приоритетов задач. В ней имеются все рассматриваемые выше функции по управлению очередями, флажками и временем. Для управления памятью ядро включает вызов программ на запрос и освобождение памятью, для управления процессами – создание процесса, установку приоритета процесса, окончание процесса. Для управления очередями реализуются функции по созданию очереди, ее открытию, считыванию из очереди, записи в очередь, задержке, диспетчеризации, удалению очереди.

Резидентные системные процессы (RSP) могут формироваться системой, составляться пользователем или быть их комбинацией. Каждый процесс RSP постоянно находится в памяти. Примерами системных процессов, введенных пользователями, могут быть программа печати оглавления файлов, распечатки файлов в шестнадцатеричной системе и т. д. Включение в эту область часто используемых системных утилит экономит время и не занимает памяти в разделах пользователей.

Генерация MP/M производится в диалоге при вызове утилиты GENSYS в соответствии с конфигурацией аппаратных средств микроЭВМ. В диалоге задается верхняя страница памяти, число консолей, использование контрольных точек при отладке, командного файла COM. Так же определяются базы сегментов памяти и включение в состав ОС системных процессов, имеющихся на диске. При этом на большинство вопросов пользователь отвечает "да" (Y) или "нет" (N). Большинство приказов MP/M совпадает с приказами CP/M (по вызову трансляторов, загрузчика, редактора и т. д.).

В связи с несколькими консолями усложняется и интерпретатор приказов. При разборе строки приказа интерпретатор прежде всего пытается отыскать очередь с тем же именем, при нахождении он записывает часть приказа в эту очередь и завершает действия. Если очередь не найдена, интерпретатор считывает с диска файл приказов FP ,обращается к ядру для выделения памяти, и если память выделена, загружает в нее код приказа и выполняет его. При отсутствии имени в файле FP поиск продолжается в командном файле COM. Интерпретатор создает также дескриптор процесса для программы и выделяет ее область стека.

Для организации взаимодействия задач, их синхронизации и исключения тупиков при распределении ресурсов в MP/M используются круговые и связанные очереди (круговая длина 2 байта, связанная – больше 2 байт). Очереди формируются ОС либо пользователем, а ядро включает функции для работы с очередями. Адаптации рассмотренных ОС применяются для работы с персональными ЭВМ.

8.4. ДИАЛОГОВАЯ МНОГОТЕРМИНАЛЬНАЯ СИСТЕМА

Общая характеристика. Диалоговая многотерминальная система (ДИАМС) предназначена для работы на СМ ЭВМ с магистральной архитектурой (СМ-4, СМ-1420 и другие программно-совместимые модели). Связь с другими ОС мини-ЭВМ типа СМ, таких, как ДОС СМ, ФОБОС, ОС РВ, может осуществляться через внешнюю память или при организации многомашинных комп-

лексов. В отличие от традиционных систем программирования входной язык ОС включает ряд специальных средств, которые существенны для виртуальной ДИАМС-машины. Функционально ОС состоит из четырех основных модулей: интерпретатора, диспетчера разделения времени, супервизора баз данных и монитора ввода-вывода (рис. 8.5). Интерпретатор обеспечивает работу на входном языке системы. Он анализирует программы на правильность, выполняя предписанные действия и выдавая при необходимости запросы другим модулям. Супервизор баз данных осуществляет отображение логической структуры данных на физическую среду – память на дисках. Монитор выполняет обмен с внешними устройствами. Диспетчер разделения времени организует работу со многими пользователями.

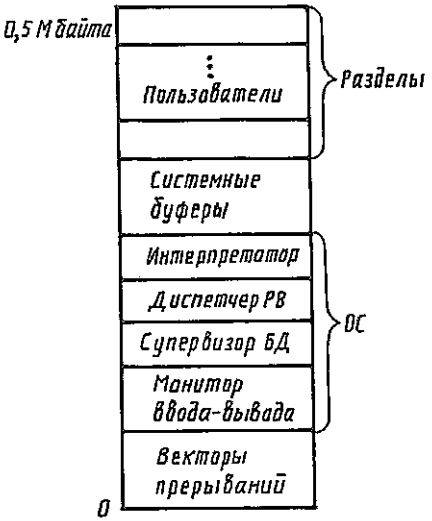


Рис. 8.5.
Распределение основной памяти в системе ДИАМС.

Входной язык включает возможности работы с адресными функциями, иерархическими древовидными структурами данных, организует виртуальное взаимодействие с внешними устройствами, обеспечивает совместную работу многих пользователей с одной или несколькими базами данных.

Во время работы система полностью находится в оперативной памяти, занимая от 30 до 55 К байт в зависимости от конфигурации аппаратуры и выбранных функциональных возможностей. Остальная память распределяется под системные буферы и разделы (до 64) пользователей (см. рис. 8.2). Раздел содержит программу пользователя, локальные переменные и данные связи с системой. Системные буферы используются для обмена с дисковой памятью и периферийными устройствами.

Интерпретатор. Он является важнейшей частью виртуальной ДИАМС-машины и позволяет пользователю работать на процедурно-ориентированном языке. Исходная программа на языке ДИАМС анализируется по предложениям и выполняется. Ускорение выполнения происходит за счет организации системы и программирования интерпретатора. Кроме того, режим интерпретации дает ряд преимуществ: проще структура системы, язык программирования обладает большей гибкостью и интерактивностью. Интерпретатор включает два режима работы: *программный* и *диалоговый*. В диалоговом выполняются команды, в программном – все программы.

Интерпретация происходит покомандно в один проход. При этом реализуется лексический и синтаксический анализ текста, вычисление выражений, присваивание переменной и т. д. Для выполнения ввода-вывода или запроса к БД происходит обращение к монитору ввода-вывода или супервизору базы данных. При достижении конца строки осуществляется либо переход на следующую строку (программный режим), либо управление передается пользовате-

лю (диалоговый). Интерпретатор является реентерабельной программой, т. е. обеспечивает повторную входимость только в моменты завершения обработки аргумента команды. Работа интерпретатора может прерваться в трех случаях: истек квант времени задачи, встретилось обращение к элементу данных на внешней памяти, распознана команда ввода-вывода. В любом случае управление передается соответственно диспетчеру разделения времени, супервизору БД, монитору ввода-вывода.

Диспетчер разделения времени. Предназначен для реализации разделения времени. При этом задания, связанные с вводом-выводом, имеют преимущества перед расчетными. Диспетчер использует несколько очередей, расположенных в таблице задания в порядке понижения приоритета. Очередь выполнения может содержать только один номер задания, которое запускается в данный момент.

Первоначально задание помещается в очередь ожидания процессора высшего приоритета, затем – в очередь приоритета и далее выполняется либо в течение кванта времени, либо до появления более приоритетного задания. Приостановленное задание по прерыванию может помещаться в очередь ожидания высшего приоритета (число выполненных команд меньше 20), среднего приоритета (число выполненных команд лежит между 20 и 4116) или низшего приоритета (число выполненных команд больше 4116).

При помещении в очередь задания просматриваются шкалы приоритетов и на выполнение выбирается первое задание из первой непустой очереди. Приостановленные задания помещаются в очередь на заданное время, после истечения которого задание заносится в очередь высшего приоритета. Для ожидания выполнения запроса к БД используются две очереди. Обращение к специальному системному заданию вывода производится через соответствующую очередь. При запуске ввода-вывода (кроме дискового) задание выбирается из очереди выполнения, после завершения операции оно ставится в очередь ожидания высшего приоритета.

Монитор ввода-вывода. Ввод-вывод производится в два этапа на логическом и физическом уровнях. По команде присвоения заданию устройства осуществляется настройка программы логического ввода-вывода на это устройство. Программа физического ввода-вывода – это обработчик прерываний. Последний ориентирован на определенный тип устройства.

Драйверы реализуют передачу информации между буфером, отводимым устройству, и регистрами устройства. Программы ввода-вывода на логическом уровне осуществляют передачу информации между буфером раздела пользователя и кольцевым буфером. При этом каждому устройству отводится буфер размером 1 К байт.

Программы логического приема отслеживают символы стирания знака и строки, приостановки и возобновления вывода и конца строки. По концу строки введенная информация передается интерпретатору для обработки команды. Программа логического вывода отслеживает символы для формирования выводимой информации. К другим функциям монитора ввода-вывода относятся управление работой внешней памяти большой емкости, организация спулинга и т. д.

Супервизор баз данных. Он осуществляет отображение логической структуры данных на физическую сферу хранения, обеспечивает физическое и логи-

ческое управление дисковой памятью. База данных имеет древовидную структуру. Для хранения данных система может использовать до восьми дисков емкостью 2, 4, 14 или 20 К байт. Вне зависимости от типа и количества используемых накопителей вся область памяти представляется в виде одного логического диска. Физически дисковое пространство разбивается на участки по 1 К байт.

Супервизор обеспечивает полную независимость между логической структурой и ее физической организацией. Логически данные представляют собой дерево, корень которого соответствует имени глобальной переменной, а узлам – индексированные переменные. Узлы дерева могут содержать как данные, так и указатели, листья – только данные. Физическая структура глобальных данных имеет вид сбалансированных деревьев, называемых B-деревьями.

Состояние блоков дисковой памяти (занят, свободен) отслеживается с помощью карт распределения памяти, которые представляют собой битовые поля (1 – блок занят, 0 – свободен).

Язык системы. Основная ориентация языка системы – создание и ведение баз данных и решение информационно-логических задач. Для этих целей подходит процедурно-ориентированный язык на основе древовидной структуры данных с набором средств текстовой обработки информации. Основным типом данных являются строки от 0 до 255 символов, под которыми выполняются арифметические, логические, строковые операции. Программа представляет последовательность команд для решения задачи. Основные группы команд включают команды создания и уничтожения переменных (SET, KILL), передачи управления (IF, DO, GO TO, FOR и т. д.), ввода-вывода (OPEN, READ, WRITE, CLOSE и т. д.), редактирования, отладки, разделения строк.

8.5. КОНЦЕПЦИИ ОС ИНМОС И UNIX

Общая характеристика. В последние годы за рубежом большое распространение для мини- и микроЭВМ получила ОС UNIX, которая имеет ряд версий, зависимых от конфигурации ВС и решаемых задач. В нашей стране создана ОС ИНМОС, в основу которой положены принципы и концепции UNIX . Благодаря использованию для написания данных ОС языка высокого уровня С системы являются легко переносимыми на различные ЭВМ. При этом разработана инструкция для создания машинно-зависимой части системы и переноса ее на другую машинную архитектуру. Инструментальность систем заключается в наличии различных программ для создания, корректировки программ пользователя, обработки файлов и текстов, построения информационно-справочных систем и т. д. Имеются средства обучения программированию. Система ИНМОС представляет собой базу, на основе которой созданы средства, расширяющие возможности ОС: программированного обучения, средства межмашинного взаимодействия, управления базами данных, базовые средства машинной графики. Данные системы работают в режиме разделения времени, используются для центров коллективного пользования, систем обучения, организации разработки прикладного и программного обеспечения.

Одно из основных достоинств системы – ее простота как для пользователя, так и для программиста. Основное решение в системе основывается на

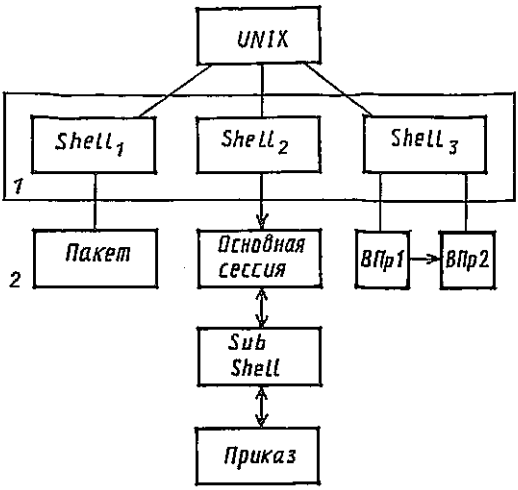


Рис. 8.6.
Иерархическое дерево системы UNIX.

трех составляющих: языке реализации С, обеспечивающем мобильность; файловой системе, унифицирующей все средства передачи информации;и командном языке (SHELL), поднимающем интерфейс пользователя с системой до уровня языка программирования (рис. 8.6). Рассмотрим принципиальные основы построения мобильной ОС.

Процессы. Процесс – единица управления и потребления ресурсов в системе. Распределение ресурсов между процессами осуществляется в ядре. При этом процессы работают в режиме "Пользователь", ядро – в режиме "Система". Ядро не является самостоятельной вычислительной единицей, поскольку программы ядра выполняются от имени процессов, их вызвавших. Выполняя прикладную программу, процесс находится в пользовательской фазе, для выполнения программы ядра он делает системный вызов. Например, давая системный вызов read (чтение файла), процесс входит в соответствующую программу ядра.

Некоторые участки системных фаз процессов являются критическими. Они необходимы для обеспечения целостности данных ядра: если процесс начал модификацию какой-либо программы, то пока она не закончится, другой процесс не может быть допущен к работе с ней. Синхронизация системных фаз процессов на границах критических участков реализуется в ядре аппаратом событий. Процессы, ждущие наступления события, снимаются с ожидания процессом, который это событие объявляет. Факт наступления события не фиксируется битом. Такая схема взаимодействия процессов соответствует механизму сопрограмм, который упрощает логику ядра системы и требует одного выделенного процесса, называемого *диспетчерским* и не имеющего пользовательской фазы.

Все процессы в системе (кроме диспетчерского) создаются с помощью операции fork . Каждый процесс имеет идентификатор, который присваивается

при его образовании, служит для работы с программой, но не связывается с ее именем. Поэтому процесс выполняет различные программы.

При начальном запуске ядром создается диспетчерский процесс с нулевым номером. В дальнейшем этот процесс загружает образы процессов в память и выгружает их на основе стратегии распределения памяти. Диспетчерский процесс порождает процесс с номером 1, который выполняет программу unit , образующую для каждого терминала системы свой процесс. С помощью интерпретатора shell пользователи теперь могут начать диалог с системой.

В системе с разделением времени (UNIX, ИНМОС) приоритет процессов не играет существенной роли. Несмотря на то что алгоритм диспетчеризации выделяет ресурс процессу с наивысшим приоритетом, все процессы имеют одинаковый приоритет, поэтому действует правило циклического квантования. В системе также отсутствуют развитые средства взаимодействия процессов. Для синхронизации используются сигналы, посылаемые между родственными процессами. Информационное взаимодействие через каналы также основано на родственных связях.

Управление памятью. В системе просто решается организация виртуальной памяти. Каждый процесс работает в своем собственном виртуальном адресном пространстве. Образ процесса включает три сегмента и стек. *Стек* используется для организации работы с подпрограммами. Сегменты, соответствующие выполняемой процессом программе, включают процедурный, динамический и сегмент данных. *Процедурный сегмент* содержит машинные инструкции и константы, *сегмент данных* и *динамический* — данные, инициируемые и не инициируемые при компиляции. *Файл* — результат работы редактора связей — содержит процедурный сегмент и сегмент данных. Место под динамический сегмент и стек выделяется при загрузке программы в основную память. Таким образом, в структуре программы нет секций. Единицей компоновки является сегмент (процедурный, данных или динамический), а единицы загрузки в виртуальную память нет вообще. Считается, что образ всего процесса помещается в виртуальное пространство. В системе не используются оверлейные структуры программы. Это связано с упрощением работы пользователя и обеспечением мобильности системы.

Для размещения программы в пределах виртуального пространства используется ряд решений. Если программа большая, ее можно разбить на части и выполнить их либо последовательно в рамках одного процесса, либо асинхронно. Однако применяется другое решение. Объем программы увеличивается не за счет команд, а за счет данных, в основном массивов. При разделении пространства для команд и данных последние могут исключаться из виртуального пространства. В системе большие массивы рассматриваются как файлы. Для предотвращения задержки обращения к файлу (прерывание, обработка системного вызова в ядре, обращение к диску, пересылка требуемой информации из памяти процесса в системный буфер) используются принципы кэш-памяти. Системные буфера объединяются в программно-реализованную кэш-память, и весь обмен между диском и памятью процессов буферизуется в этой памяти. Всякий раз, когда нужен блок данных диска, проверяется его наличие в кэш-памяти, если он находится там, обращения к диску не происходит.

Файловая система. Она не является надстройкой над ОС. Наоборот, ее понятия первичны и создают основу для других системных понятий. Файловая

структура известна ядру системы, механизм работы с файлами встроен в ядро и составляет его часть.

Файловая система имеет иерархическую структуру, называемую *деревом*. В узлах дерева расположены каталоги, содержащие перечень других файлов и каталогов, к которым можно обратиться из данного. Корнем дерева служит каталог, а листьями – только файлы. Каталоги тоже являются файлами, но запись в них может делать только ядро системы. В такой структуре путь в дереве связывается с именем файла. Полным именем файла является последовательность имен каталогов, стоящих на пути от корня дерева к этому файлу. Трудности вызывает то, что от корня к файлу может вести не один путь, т. е. файл может иметь несколько разных имен. Многократное использование одного и того же файла возможно, но это происходит редко, поэтому файловую структуру можно считать деревом. Полные имена в системе неудобны из-за их длины и необходимости просматривать все каталоги на пути к диску. Так возникает понятие текущего каталога, по которому файл имеет простое имя.

Каждый диск содержит файловую структуру. Операция монтирования позволяет объединить деревья в общее дерево файлов. После запуска системы имеется дерево файлов на системном диске. По мере монтирования томов к основному дереву подключаются поддеревья. Монтирование – это выделение файла в существующем дереве и объявление его корневым каталогом подсоединяемого дерева.

Листья дерева могут выступать как специальные файлы, под которыми понимаются внешние устройства. Специальные файлы обрабатываются как обычные и каталоги. В системе различают два вида специальных файлов: байт-ориентированные и диск-ориентированные. Блок-ориентированный интерфейс дает возможность обмениваться с дисками и лентами поблочно. Байт-ориентированный интерфейс позволяет обмениваться со всеми внешними устройствами, рассматривая их как массивы байтов. Байт-ориентированный интерфейс для лент и дисков называется прозрачным вводом-выводом и позволяет передавать порции информации больше одного блока.

Командный язык. Он называется по имени интерпретатора shell и является одним из преимуществ системы. В нем синтезированы достоинства языков управления заданиями пакетных систем, командных языков интерактивных систем, а также современных языков программирования. Внешне командный язык – это язык программирования, позволяющий использовать переменные, различные операторы управления, группировать операторы в более сложные конструкции. Программу, написанную на командном языке, выполняет интерпретатор shell.

Интерпретатор не знает списка всех команд, поскольку имя команды должно быть именем файла. Shell находит этот файл и организует его выполнение. Однако существуют встроенные команды, выполняющиеся самим интерпретатором. Это связано с теми действиями, которые запрашивает интерпретатор. Например, команда wait приостанавливает диалог до завершения всех асинхронных процессов, "отцом" которых является сам shell, поэтому она может выполняться только интерпретатором.

Для выполнения файла shell порождает процесс, который и будет его выполнять. Перед этим интерпретатор производит анализ типа файла: выполняе-

мый или текстовой. Первый является обычной программой, а второй – командным файлом. В первом случае порожденный процесс будет выполнять указанный файл, во втором – порожденный процесс должен выполнять shell, интерпретирующий командный файл. Набор команд можно расширить, написав программу на языке программирования или языке SHELL. Нужно, однако, иметь в виду, что расширение команд за счет языка SHELL – не очень эффективный путь с точки зрения времени выполнения, поскольку язык является интерпретирующим, а не компилирующим.

Для работы с системой UNIX пользователь имеет в своем распоряжении более сотни различных сервисных программ (утилит). Порядок работы с системой UNIX подробно описан в работе [10].

8.6. ПАКЕТЫ ПРОГРАММ ДЛЯ ТЕЛЕОБРАБОТКИ, АВТОМАТИЗАЦИИ И ПРОЕКТИРОВАНИЯ

Основные пакеты ПО СМ ЭВМ. *Системa телеобработки данных (СТД)* предназначена для управления процессами передачи данных по линиям связи, обслуживания систем дистанционной обработки данных, концентраторов и мультиплексоров. Система имеет широкий диапазон областей применения от научных исследований до систем управления производством. На базе СТД могут создаваться системы дистанционной пакетной обработки и системы обмена между различными ЭВМ.

Пакет программ состоит из следующих модулей: монитора телеобработки данных, программ обслуживания терминалов, программ обслуживания ввода-вывода, связанных с управлением модемом и передачей данных. Пользователю система предоставляет возможности инициализации линий передачи данных, прием и передачу сообщений по линиям связи, буферизацию сообщений, синхронизацию при передаче данных, связь системных программ с пользовательскими.

Система ведения банка данных ИРИС используется в информационно-поисковых системах, системах обработки статистических данных, системах оперативного учета и планирования, АСУ ТП, системах сбора и обработки экспериментальных данных. Доступ к банкам данных, созданным под управлением ИРИС, осуществляется в пакетном и диалоговом режимах. Предусмотрено общение с пользователем на языке, близком к естественному. Кроме того, в системе ИРИС имеется возможность подключить программы, составленные на языках ассемблера М-4030, макроассемблера СМ, ФОРТРАН, КОБОЛ. Система является дальнейшим развитием системы интегрированной обработки данных (СИОД), которая полностью с ней совместима. Применение системы ИРИС основывается на следующих особенностях: возможности накопления большого объема данных произвольной структуры; возможности быстрого доступа к данным через локальные или удаленные терминалы; возможности получения данных из БД по одному или нескольким признакам; использовании для написания пользовательских программ как ассемблера, так и языков высокого уровня; наличии средств, повышающих надежность функционирования системы (защитные копии, контрольные точки, повторный запуск); разнообразии форматов данных.

Пакет *программ методов оптимизации* предназначен для расширения воз-

можностей ДОС в области решения задач оптимизации методом линейного и динамического программирования. Пакет представляет собой библиотеку модулей, написанных на ФОРТРАН-IV. Модули обеспечивают решение общей задачи линейного программирования, транспортной задачи, задачи целочисленного программирования, одномерной задачи динамического программирования. Задачи линейного программирования решаются симплекс-методом, целочисленного программирования – методом ветвей и границ. Транспортная задача решается венгерским методом. Модули решения задачи динамического программирования обеспечивают решения одномерной задачи с аддитивным критерием качества.

Пакет программ имитационного моделирования позволяет осуществлять имитационное моделирование непрерывных, непрерывно-дискретных и дискретных процессов. Область пакета практически не ограничена, поскольку любые и система, и процесс могут быть сведены к одной из трех названных групп без особых нарушений требований адекватности явлений и модели. К этим группам относятся системы обработки и передачи данных, класс задач, представляемый модулями массового обслуживания.

Пакет представляет набор модулей, написанных на языке ФОРТРАН-IV с собственной управляющей программой. Настройка пакета на требования пользователя осуществляется в процессе генерации пакета в режиме диалога. Модули пользователя, описывающие модуль, пишутся на языке ФОРТРАН-IV. При этом допускается включение программ, написанных на макроассемблере. Пакет является открытой системой, работает под управлением ОС ФОБОС и требует 56 К байт памяти.

Пакет программ обработки данных эксперимента предназначен для расширения возможности ДОС СМ ЭВМ в области анализа и обработки экспериментальных данных. Комплекс модулей пакета позволяет реализовать следующие функции: анализ данных планируемого эксперимента, а также вычисление основных и непараметрических статистик; прогнозирование и анализ временны́х рядов; генерацию случайных чисел; вычисление специальных функций и многомерный анализ; операции над матрицами. В состав пакета входят также сервисные модули. Все модули пакета написаны на языке ФОРТРАН-IV и не содержат операций ввода-вывода. Пакет является открытой системой и может расширяться пользователем.

Программное обеспечение автоматизации проектирования. На базе СМ ЭВМ серийно выпускаются несколько типов вычислительных комплексов, предназначенных для автоматизации проектирования. Одним из них является автоматизированное рабочее место конструктора радиоэлектронной аппаратуры (АРМ-Р). Этот комплекс включает широкий набор устройств и средств машинной графики для обработки текстовой и графической информации. В состав средств машинной графики входят алфавитно-цифровой и графический дисплеи, графопостроители, устройства ввода графической информации.

Базовое программное обеспечение АРМ включает ДОС АРМ, тестовую систему ТЕСАРМ, базовую оперативную графическую систему, вспомогательные программы и систему подготовки данных для станков с числовым программным управлением. Базовое программное обеспечение АРМ предоставляет пользователю следующие возможности: решать до 5 различных задач, вводить и просматривать введенные чертежи с их корректировкой, рассматривать лю-

бые фрагменты чертежей в увеличенном масштабе, создавать архивы описаний чертежей и обращаться к ним по именам, вводить и редактировать текстовые материалы, получать результаты проектирования.

Для многих классов задач, особенно проектирования и конструирования, необходимо, чтобы процесс взаимодействия человека и системы осуществлялся на основе графических изображений. Возможности такого взаимодействия предоставляются благодаря наличию графического дисплея, графопостроителя и средств машинной графики. Последняя позволяет с помощью светового пера идентифицировать часть изображений, выбирать операцию из нескольких (построение линии, окружности, дуги и т. д.), выбирать объект из множества объектов (набор стандартных графических изображений), получать новый объект путем перемещения светового пера (вычерчивание линий от руки на экране дисплея). При решении задач типа размещения элементов на плоскости последние записываются в библиотеку. Размещение иллюстрируется изображением на экране дисплея: задач, решение которых носит итеративный характер (трассировка печатной платы, задачи оптимизации); задач, связанных с внесением изменений в чертежи или текстовые материалы; подготовки и отладки прикладных программ. Дальнейшее развитие АРМ направлено на разработку аппаратных и программных средств. В частности, создано групповое АРМ-2, включающее центральный комплекс на базе СМ-4 и терминальные станции, подключаемые локально или удаленно. Программное обеспечение группового АРМ включает ОС РВ, драйверы для подключения устройств к системе, программные средства для управления работой терминалов и пакеты прикладных программ, ориентированных на конкретный тип проектирования и класс решаемых задач.

Средства машинной графики позволяют осуществлять следующие функции: обслуживание запросов графических терминалов, передачу для выполнения в выделенные программные разделы; организацию базы данных для поиска, чтения, записи информации; формирование и обработку библиотечных данных; преобразование графической информации для редактирования и обработки графики в режиме диалога.

Дальнейшее развитие АРМ связано с использованием базовых средств персональных ЭВМ с объединением их в локальные сети.

В последнее время для комплексов мини-ЭВМ, включающих микроЭВМ, получило широкое распространение использование ОС реального времени с разделением функций РАФОС и пакета машинной графики ГРАФОР. РАФОС позволяет строить системы, сочетающие решение задач реального времени с многопользовательской работой в режиме СРВ. Система ориентирована на применение в АСУ ТП и в комплексах автоматизации научных исследований. В составе РАФОС можно выделить управляющую систему, драйверы внешних устройств, файловую систему, мониторы, системные программы и библиотеки, средства программирования.

Выводы

1. Организация вычислительных процессов в мини- и микроЭВМ обеспечивается широким набором ОС и пакетов программ в зависимости от использования в конкретной проблемной области.

2. В зависимости от конфигурации аппаратных средств и области применения в СМ ЭВМ может использоваться более десятка типов ОС. Эти ОС делят на системы общего назначения, реального времени, разделения времени; по типу исходного носителя – на перфоленточные и дисковые, по числу пользователей – на однопользовательские и многопользовательские.

3. Организация вычислений в микроЭВМ в однопрограммном режиме поддерживается ОС СР/М, включающей ядро, базовую систему ввода-вывода и интерпретатор приказов. Многопрограммная работа поддерживается ОС МР/М, которая на уровне ядра включает ряд функций для работы с параллельными процессами.

4. Организация вычислительных процессов в многотерминальной ОС ДИАМС обеспечивается интерпретатором команд, монитором ввода-вывода, супервизором баз данных и диспетчером разделения времени.

5. Основные системные решения в ОС UNIX базируются на трех составляющих: языке С, обеспечивающем мобильность системы; файловой системе, унифицирующей все средства обработки информации; командном языке, поднимающем интерфейс пользователя до уровня языка программирования.

6. Функции ОС в микро- и мини-ЭВМ расширяются за счет пакетов общего назначения. К ним относятся пакеты для поддержания телеобработки, ведения баз данных, оптимизации, имитационного моделирования, обработки эксперимента, автоматизации проектирования, работы с текстами, машинной графики.

Рекомендуемая литература: [5, 10, 19, 22, 27].

Вопросы для контроля

1. Охарактеризуйте программное обеспечение СМ ЭВМ.
2. Дайте характеристики наиболее распространенных ОС микроЭВМ.
3. Определите назначение компонентов СР/М.
4. Как усложняются функции в МР/М по сравнению с СР/М?
5. В чем функциональное назначение ОС ДИАМС?
6. Назовите и поясните достоинства системы UNIX.
7. В чем отличие управления процессами в системе UNIX?
8. Определите особенности управления памятью в системе UNIX.
9. Охарактеризуйте командный язык UNIX.
10. Определите основные направления пакетов программ СМ ЭВМ.
11. Охарактеризуйте организацию процессов в АРМ-Р.

9. АВТОМАТИЗАЦИЯ ПРОЕКТИРОВАНИЯ ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА

9.1. ПОНЯТИЕ О ФОРМАЛИЗАЦИИ И МОДЕЛИРОВАНИИ ФУНКЦИОНИРОВАНИЯ ВС

Проектирование. Под *проектированием вычислительного процесса* понимается процесс составления описания, необходимого для создания еще не существующего объекта (алгоритма его работы или алгоритма процесса), который осуществляется преобразованием первичного описания, оптимизацией заданных характеристик в детализированное описание. Проектирование вычислительных процессов ВС (ее компонентов) может включать: постановку задачи, составление формального описания ВС с получением модели, программирование этой модели, ее испытание и принятие решений по результатам испытаний.

Основным средством автоматизации проектирования вычислительного процесса в настоящее время является моделирование на основании различных подходов: аналитических методов, имитационного описания и сетей Петри. В свою очередь средства автоматизации проектирования могут включать математическое, программное и языковое обеспечение. Под математическим обеспечением понимается совокупность методов и моделей для описания вычислительных процессов. Программное обеспечение включает прикладные программы моделирования и системные программы (трансляторы с языков моделирования). Языковое описание содержит языковые средства описания моделей вычислительных процессов. Рассмотрим средства формализации вычислительных процессов и языки для их описания.

Один из методов определения параметров ВС и организации вычислительного процесса – использование аналитических моделей. Последние основываются на математических уравнениях, описывающих процесс функционирования системы. Достоинством этих моделей является простота. Они дают возможность получить оценки параметров ВС в широком диапазоне варьирования переменных, что важно на этапе разработки и проектирования. В то же время аналитические модели имеют невысокую точность (30–40 %) по сравнению с реальными характеристиками, а получаемые аналитические модели не всегда могут быть решены в общем виде, что требует более упрощенного представления реальной ВС.

Другой метод определения параметров ВС и проектирования вычислительного процесса – применение имитационных программных (или аппаратных) моделей. При этом структура и процесс обработки информации в ВС воспроизводится на основе программ (аппаратурных структур). Статистические характеристики функционирования системы определяются на основе метода Монте-Карло. В последнее время большое распространение для моделирования процессов в ВС получили сети Петри.

Анализ соответствия проектируемой ВС задаваемым требованиям может производиться на примере группы задач, решаемых в системе. При этом обобщение описания класса задач и данных достигается при учете статистических закономерностей поступления и обработки информации в ВС. Поэтому ВС, как правило, на системном уровне рассматривают в виде системы массового обслуживания (СМО).

Основные понятия СМО. Система, рассматриваемая как СМО, состоит из обслуживающих статических объектов-ресурсов, которыми могут быть ЭВМ, отдельные устройства ЭВМ, программные комплексы и т. д. В СМО, кроме объектов статического типа, присутствуют динамические объекты, называемые *заявками* или *транзактами.*

Функционирование СМО рассматривается как процесс прохождения заявок через систему. При этом заявка может поступить на обслуживание в ресурс. Тогда переменная v, характеризующая состояние ресурса, примет значение "Занято", а переменная w, характеризующая состояние заявки, – "Обслуживание". Если ресурс не занят обслуживанием, то переменная v принимает значение "Свободен". При поступлении заявки на вход занятого ресурса она становится в очередь. Переменная w заявки в очереди имеет состояние "Ожидание". Переменные v и w принимают два состояния и являются булевыми переменными. Состояние ресурса характеризуется еще переменной r – длиной очереди заявок на входе (число заявок). Правило, по которому заявки поступают из очереди на обслуживание в ресурсы, называется *дисциплиной обслуживания.* С заявкой может быть связана величина права ее на обслуживание. Приоритетом называется число, определяющее порядок обслуживания заявки.

В СМО используются различные дисциплины обслуживания. При равенстве всех приоритетов применяется простейшая дисциплина обслуживания – бесприоритетная. Среди бесприоритетных наиболее часто употребляемая дисциплина FIFO. В дисциплине LIFO заявки выбираются из конца очереди. При различных приоритетах заявок на входе ресурса образуется несколько очередей, каждая для заявок со своим приоритетом. Заявки из очередей с низким приоритетом обслуживаются, если нет заявок в более привилегированных очередях. Приоритеты могут быть статическими и динамическими в зависимости от изменения приоритета в процессе прохождения заявки. Возможны дисциплины обслуживания как с ожиданием конца начатого обслуживания, так и с его прерыванием при поступлении заявки с более высоким приоритетом на занятый ресурс.

Любое изменение состояния системы (изменение переменных v, w, r) называется *событием.* Считается, что события происходят мгновенно в дискретные моменты времени. Для анализа СМО необходимы сведения о составе элементов и связи их друг с другом; о дисциплине обслуживания каждого ресурса, значении параметров ресурса (внутренние параметры, входящие в вектор X), значении параметров входных потоков заявок (внешние параметры, входящие в вектор Q). В результате анализа вычисляются значения выходных параметров СМО, входящих в вектор Y. К выходным параметрам относятся: производительность системы – среднее количество обслуживаемых заявок в единицу времени; вероятность обслуживания заявки (для заявок, которые могут покидать систему необслуженными в случае превышения заданного вре-

мени нахождения в очереди); коэффициенты загрузки оборудования; средние величины очередей к ресурсам и т. д.

Если задана структура СМО, то вектор выходных параметров является функцией векторов внутренних и внешних параметров в виде $Y = F(X, Q)$. В этом случае считается, что задана аналитическая модель СМО.

Средства автоматизации моделирования. Для моделирования на ЭВМ структуры ВС или ее отдельных компонентов нужен аппарат моделирования, в котором должны быть предусмотрены способы организации данных, удобные средства формализации и воспроизведения динамических свойств модели, а также возможность описания стохастических систем, т. е. процедуры анализа и генерирования случайных чисел. В состав данных средств автоматизации моделирования ВС входят один или несколько языков описания объектов моделирования, средств обработки языковых конструкций – транслятор, система выполнения имитационного процесса во времени и набор программ для организации модельных экспериментов.

Применение универсальных языков программирования представляет большие возможности для моделирования, однако требует значительных усилий, затрачиваемых на программирование. Поэтому для описания формальных процессов ВС используют специализированные языки имитационного моделирования. Последние делятся на две группы, соответствующие непрерывным и дискретным процессам.

По способу реализации различают системы моделирования, использующие универсальные системы программирования, и специальные системы. В первом случае система моделирования расширяет систему программирования. Обычно в качестве базового языка программирования используется АЛГОЛ, ФОРТРАН, ПЛ/1. На базе АЛГОЛа строятся языки моделирования ASPOL, SOL, SIMULA, СЛЭНГ. На базе ФОРТРАНа реализованы языки SMPL, GASP, SIMSCRIPT, на базе ПЛ/1 – PLSIM, MPL/1. К специальным языкам моделирования процессов и систем относятся GPSS, GSL, ACUM и др.

Независимо от базовой системы программирования, имеются три типа языков моделирования ВС, различающиеся способом описания в имитационной модели реальных процессов: организующие взаимодействие транзактов (GPSS), управляющие процессами (ASPOL, SIMULA, PLSIM), составляющие расписание событий (GASP, SIMSCRIPT, SMPL).

Модели программных компонентов ВС. При моделировании внешней средой служат прикладные программы и данные, внутренней средой – устройства ЭВМ и ОС. С точки зрения степени отображения взаимодействия отдельных компонентов модели программы делятся на несколько типов. К первому типу относятся модели смеси, отображающие процентный состав операций в моделируемом алгоритме (статистика Гибсона и др.), ко второму – марковские модели, учитывающие связь команд в программе (язык PACSS). К третьему типу относятся модели алгоритмов программ, отображающих их структуру (циклы, переходы и т. д.). Четвертый тип определяет модели, которые отражают результаты прогона программ на одном из наборов данных. Однако они громоздки.

Элементами модели программы могут быть задание, шаг задания, макрокоманда или псевдокоманда. Общецелевые системы моделирования позволяют строить простые вероятностные модели программ, а при точном модели-

рования требуют больших затрат. Модели программ на уровне псевдокоманд используются обычно для описания компонентов ОС.

Для задания потоков внешних данных нашли применение вероятностные и марковские модели. При представлении логических структур данных, обрабатываемых программами, и имитации их отображения во внешней и основной памяти в языки моделирования встраиваются модели структур данных (запись, файл, массив), средства отображения их на внешнюю память – цилиндр, дорожку (GSS-II), буферный пул (QSSL) и т. д.

9.2. АНАЛИТИЧЕСКИЕ МЕТОДЫ И МОДЕЛИ

Аналитические методы. При использовании аналитических методов строится математическая модель системы, представляющая ее свойства в виде математических объектов и отношений, например в виде дифференциальных или интегральных уравнений. При этом требуемые зависимости выводятся из модели путем применения математических правил. Модель может строиться на основе понятий символики и методов некоторой теории, например теории массового обслуживания. В последнем случае необходимо построить модель вычислительного процесса (в простейшем случае на основе марковской модели) и определить параметры функционирования СМО. В свою очередь СМО определяется совокупностью трех характеристик: входного потока заявок, параметров обслуживающих узлов, дисциплин обслуживания.

Рассмотрим отдельные моменты описания в аналитической форме функционирования ВС. Для построения моделей вычислительных процессов прежде всего необходимо определять трудоемкость моделирующих алгоритмов, т. е. количество вычислительной работы, требуемой для реализации алгоритма. В первом приближении трудоемкость алгоритма можно охарактеризовать совокупностью параметров: – θ – среднее количество процессорных операций, выполняемых за одну реализацию алгоритма; $N_i \{i = \overline{1,k}\}$ – среднее количество обращений к файлам $F_i\{i = \overline{1,k}\}$ за одну реализацию алгоритма; $\theta_i\{i = \overline{1,k}\}$ – среднее количество байтов, передаваемых за одно обращение к файлам $F_1, \ldots, F_k$ соответственно. Таким образом, значение θ характеризует трудоемкость счета, а $N_i, \theta_i\{i = \overline{1,k}\}$ – трудоемкость процесса ввода-вывода информации.

Модели процессов. Наибольший интерес представляют те детали процессов, которые характеризуют использование ресурсов системы. С учетом сведений о трудоемкости алгоритма и возможных подходов к анализу и синтезу ВС к моделям предъявляются следующие требования: определение порядка порождения алгоритмов запросов на каждый вид обслуживания (счет, ввод-вывод); определение трудоемкости обслуживания запросов (количество операций процессора при счете и байтов при вводе-выводе); отображение вычислительных процессов как реализация случайного процесса (случайный характер запросов и характеристик трудоемкостей запросов); соответствие реальным процессам с точностью до математических ожиданий их одноименных характеристик.

На основании сформулированных требований к модели рассмотрим вычислительный процесс как последовательность этапов счета и ввода-вывода

при обращении к файлам. Состояние процесса счета обозначим S_0, а состояния, соответствующие обращениям к файлам, – $S_i\{i=\overline{1,k}\}$. Вычислительный процесс – это последовательность состояний $S_{ti}\{i=\overline{1,M}\}$, причем $S_{ti}\in\{S_0,...,S_k\}$, а последнее состояние процесса S_{k+1}.

Марковская модель. Наиболее простую модель можно получить, если принять, что последующие состояния вычислительного процесса зависят от текущего состояния и не зависят от предыдущих. Тогда вычислительный процесс становится *марковским*, определяемым множеством его состояний $\{S_0, ..., S_{k+1}\}$, матрицей вероятностей переходов $P=[p_{ij}]i,j=\overline{0,k+1}$ и распределением вероятностей $a_i\{i=\overline{1,k+1}\}$ состояний $S_0,...,S_{k+1}$ в момент времени 0. Элементы p_{ij} матрицы P определяют вероятность перехода процесса из состояния S_i в состояние S_j. Матрица P – стохастическая, построчные суммы элементов которой $\sum_j p_{ij}=1$.

Будем считать, что вычислительный процесс начинается с состояния S_0, т. е. с этапа счета. Этап ввода-вывода может инициироваться процессором. Это означает, что после ввода-вывода следует этап счета. В таком случае вероятности начальных состояний $(a_0, a_1, ..., a_{k+1}) = (1, 0, ..., 0)$ и матрица вероятностей переходов

$$P=\begin{array}{c} \\ S_0 \\ S_1 \\ \vdots \\ S_{k+1} \end{array}\begin{array}{c} \begin{array}{cccc} S_0 & S_1 & \dots & S_{k+1} \end{array} \\ \begin{bmatrix} 0 & P_{0,1} & \dots & P_{0,k+1} \\ 1 & 0 & \dots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \end{array}$$

Из состояний счета S_0 процесс с соответствующей вероятностью может перейти в состояния $S_i\{i=1,k\}$ или в конечное состояние S_{k+1}. Из состояний $S_i\{i=1,k\}$ процесс с вероятностью 1 возвращается в состояние счета S_0. Значения вероятностей $p_{ij}\{i=0, j=\overline{1,k+1}\}$ предопределяют ход вычислительного процесса и зависят от параметров трудоемкости алгоритма. Трудоемкость алгоритма определяет среднее число $N_1, ..., N_k$ обращений к файлам $F_1,...,F_k$. Следовательно, число переходов из S_0 в S_i должно быть $\sum_{i=1}^{k} N_i$, а с учетом перехода в состояние S_{k+1} вычислительный процесс должен выходить из состояния S_0 в среднем $\sum_{1}^{k} N_i + 1$ раз.

Количество работ, выполняемых на каждом из этапов, характеризуется параметрами $\theta_i\{i=1,k\}$ алгоритма. Тогда средняя трудоемкость этапа счета $\theta_0=\theta/N$, причем трудоемкость каждого этапа рассматривается как случайная величина ϑ_i с математическим ожиданием $\theta_i\{i=0,k\}$.

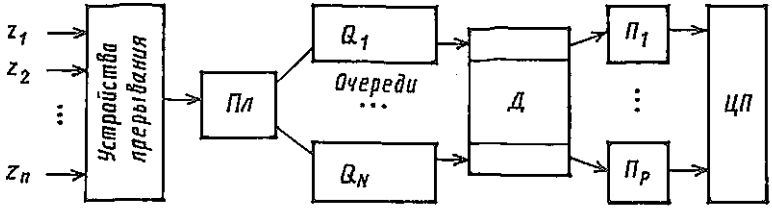


Рис. 9.1.
Обработка заявок в системе реального времени.

Пример. Построим марковскую модель вычислительного процесса, порождаемого алгоритмом, трудоемкость которого характеризуется параметрами: $\theta = 10$ млн операций; $N_1 = 19$ обращений к файлу F_1; $N_2 = 80$ обращений к файлу F_2; $\theta_1 = 2000$ байт, $\theta_2 = 500$ байт за обращение к файлу. Среднее число этапов счета при одном прогоне алгоритма $N = N_1 + N_2 + 1 = 100$. Вероятности переходов процесса из состояний счета S_0 в состояния S_1, S_2, S_3 равны соответственно $p_{0,1} = N_1/N = 0{,}19$; $p_{0,2} = N_2/N = 0{,}8$; $p_{0,3} = N_3/N = 0{,}01$.

Средняя трудоемкость этапа счета $\theta_0' = \theta/N = 0{,}1$ млн операций.

Обслуживание заявок. Для управления различными объектами в реальном масштабе времени используются управляющие ЭВМ, которые реализуют совокупность программ $П_i \{i = \overline{1,p}\}$. В объекте могут генерироваться заявки $z_i \{i = \overline{1,\omega}\}$, инициирующие выполнение программ $П_1, ..., П_\omega$ соответственно. При наличии одного процессора эти программы могут выполняться последовательно, поэтому возникают очереди заявок. Заявкой может быть и требование ресурса ВС.

Обработка заявок организуется по схеме (рис. 9.1). Заявки z_i поступают в устройство прерывания, которое приостанавливает ЦП, и вызывается планировщик Пл. Последний определяет тип заявки и ставит ее в соответствующую очередь на обслуживание $(Q_1, ..., Q_N)$. Процедура выбора заявок из очередей выполняется диспетчером Д, который реализует дисциплины обслуживания заявок и вызова программ $П_k$ $(k = \overline{1,p})$. После обслуживания считается, что заявка покидает систему. По окончании программы $П_k$ управление вновь передается диспетчеру. Если очереди пусты, процессор переводится в состояние "Ожидание".

Характерная особенность рассматриваемой системы – наличие предельных ограничений на время пребывания заявок в системе. Это время состоит из времени ожидания в очереди и обслуживания. Для упрощения примем, что процессор – единственный ресурс, необходимый для обслуживания заявок.

Функционирование системы реального времени рассматривается как совокупность трех процессов: поступления заявок на вход, диспетчеризации и обслуживания заявок. Время пребывания заявки зависит от параметров каждого из этих процессов. Поскольку функционирование системы носит стохастический характер, ее можно описывать в терминах теории массового обслуживания.

Потоки заявок. В общем случае поток заявок рассматривается как случайный процесс, задаваемый функцией распределения промежутков времени между моментами поступления двух соседних заявок. Поток характеризу-

ется интенсивностью λ (число заявок, поступающих в единицу времени). Простейший поток является стационарным ($\lambda = \text{const}$), заявки поступают независимо, по одной в единицу времени, функция распределения для этого потока $p(\tau) = 1 - e^{-\lambda\tau}$. Для него число заявок, поступивших в систему за время t, распределено по закону Пуассона:

$$P_r(k, \tau) = \frac{(\lambda k)^k}{k!} e^{-\lambda\tau} (\lambda > 0),$$

где P_r – вероятность поступления за время τ k заявок.

В некоторых системах интервалы времени между поступлением заявок независимы. Заявки образуют рекуррентный поток (поток с ограниченным последействием). Простейший поток является частным случаем рекуррентного. Примером рекуррентного потока является поток Эрланга k-го порядка, у которого интервалы времени между моментами поступления двух последовательных заявок представляют собой сумму k независимых случайных величин, распределенных одинаково по показательному закону с параметром λ.

Наибольшее число аналитических результатов получено для простейшего потока, поскольку другие виды потоков при анализе приводят к усложнению математических выкладок и громоздким результатам. Поэтому аналитическое исследование систем реального времени целесообразно проводить в предположении о простейшем потоке заявок, для чего достаточно задать интенсивности их поступления и рассмотреть время обслуживания.

Длительность обслуживания заявок. В общем случае эта длительность – случайная величина с законом распределения $B(\tau)$ и математическим ожиданием ϑ. Длительность обслуживания заявки процессором определяется временем выполнения соответствующей программы. В случае малой разветвленности это время постоянно и равно θ. В случае большого числа разветвлений время выполнения программы рассматривается как случайная величина с математическим ожиданием ϑ и дисперсией D, которые могут вычисляться на основе марковской модели процесса или путем статистической обработки многих прогонов программы.

При известных математическом ожидании и дисперсии программы время ее выполнения может аппроксимироваться гамма-распределением с плотностью вероятности

$$b_k(\tau) = ((k/\vartheta)^k/\Gamma(k)) \exp(-k\tau/\vartheta_{\tau^{k-1}}), \tau > 0,$$

где ϑ – математическое ожидание длительности обслуживания; k – параметр распределения ($k \geqslant 1$); $\Gamma(k)$ – гамма-функция.

При $k = 1$ получаем экспоненциальное распределение. При целочисленном k $\Gamma(k) = (k-1)!$, и получается распределение Эрланга k-го порядка. Если известно только среднее время выполнения программы и отсутствуют сведения о законе распределения, то время выполнения программы целесообразно аппроксимировать экспоненциальным распределением вида $b(\tau) = (e^{-\tau/\vartheta})/\vartheta$.

Характеристики дисциплин обслуживания. Различают бесприоритетное обслуживание заявок и с приоритетами, которые могут быть относительными, абсолютными и смешанными. При бесприоритетном обслуживании заявок ис-

пользуются три стратегии: в порядке поступления (FIFO), в обратном порядке (LIFO), путем случайного выбора из очереди. Эти дисциплины характеризуются одинаковым временем ожидания заявок, но дисциплина FIFO минимизирует дисперсию времени ожидания, поэтому она используется чаще. Если в систему поступают заявки M типов с интенсивностями λ_i и каждый из потоков пуассоновский, известны математическое ожидание ϑ_1, ..., ϑ_M и вторые начальные моменты $\vartheta_1^{(2)}$, ..., $\vartheta_M^{(2)}$ времени обслуживания заявок типа $1, M$, то среднее время ожидания заявок всех типов равно $\omega = \sum_{i=1}^{M} \lambda \vartheta_i^{(2)} / (2(1-R))$, где $R = \sum_{i=1}^{M} \rho_i$ – суммарная загрузка системы $(R < 1)$, $\rho_i = \lambda_i \vartheta_i$.

Если отдельным типам заявок присвоить убывающие приоритеты в виде чисел 1, 2, ..., n, учитывающиеся только в момент выбора заявки на обслуживание, то эти *приоритеты относительные.*

Если в систему поступает M простейших потоков с интенсивностями λ_1, ..., λ_M и длительности обслуживания заявок каждого потока имеют математические ожидания ϑ_1, ..., ϑ_M и вторые моменты $\vartheta_1^{(2)}$, ..., $\vartheta_M^{(2)}$, то среднее время ожидания заявок с приоритетами $k = 1, ..., M$ определяется следующим образом:

$$\omega_k = \sum_{i=1}^{M} \lambda_i \vartheta_i^{(2)} / (2(1 - R_{k-1})(1 - R_k)),$$

где R_{k-1} и R_k – загрузки, создаваемые потоками заявок z_1, ..., z_{k-1} и z_1, ..., z_k соответственно.

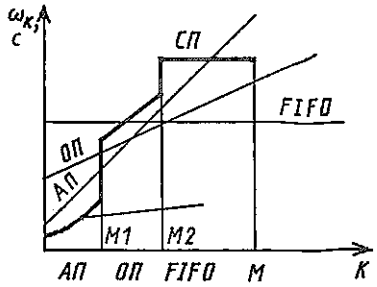
При относительном приоритете не допускается прерывания заявки, захватившей процессор, более высокоприоритетной заявкой. Дисциплина обслуживания, при которой высокоприоритетная заявка может прервать заявку с меньшим приоритетом, называется *дисциплиной обслуживания с абсолютными приоритетами.* Среднее время ожидания для заявки с абсолютным приоритетом и параметрами, аналогичными вышерассмотренным,

$$\omega_k = R_{k-1} \vartheta_k / (1 - R_{k-1}) + \sum_{i=1}^{M} \lambda_i \vartheta_i^{(2)} / (2(1 - R_{k-1})(1 - R_k)),$$

где R_j – загрузка системы от первых j потоков заявок.

Эффект от использования абсолютных приоритетов иллюстрируется на рис. 9.2, на котором ОП – график относительного приоритета; АП – абсолютного. Время ожидания заявок с высокими приоритетами уменьшается, а с низкими увеличивается. Чтобы выполнить ограничения по всем видам заявок, можно одним заявкам присвоить относительные, другим – абсолютные приоритеты, а остальные заявки обслуживать без приоритетов (*смешанные приоритеты*). Тогда время ожидания в системе со смешанными приоритетами (СП) изменится. При задании смешанной дисциплины обслуживания произвольного

Рис. 9.2.
Время ожидания в системе с различными приоритетами.



вида, когда один и тот же тип заявок может быть бесприоритетным, с относительным и абсолютным приоритетами по отношению к различным группам других заявок, формальное описание производится с использованием матрицы приоритетов. Последняя отражает вид и степень приоритетности между потоками заявок.

Получение аналитических моделей возможно в частных случаях: входные потоки стационарны, ординарны, с отсутствием последствия; время обслуживания заявок распределено по экспоненциальному закону; может использоваться один вид дисциплин обслуживания. Такой подход позволяет получить предварительные оценки функционирования ВС, например оценки влияния конфликтных ситуаций в оперативной памяти и оценки влияния общих ресурсов ОС на системную производительность многопроцессорной ВС. Эти предварительные оценки на 15–20 % в среднем отличаются от результатов, полученных экспериментально.

9.3. АППАРАТ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

Принцип имитационной модели. При моделировании процессов не обязательно преобразовывать математическую модель в специальную систему уравнений относительно искомых величин. Иногда достаточно имитировать сами явления с сохранением их логической структуры, временны́х соотношений с использованием ЭВМ. В противоположность аналитическим и численным методам содержание операций, осуществляемое при имитационном моделировании, мало зависит от выбора величин в качестве исходных переменных. При имитационном моделировании процессов отдельные величины можно задавать точно (быстродействие, объем памяти и т. д.), другие приближенно, используя вероятностное описание (величина очереди на обслуживание заявок к устройствам, время появления заявок и т. д.). При этом результаты, полученные при воспроизведении на имитационной модели исследуемых процессов, являются результатами случайных событий, поэтому для нахождения объективных характеристик процесса требуется многократное воспроизведение со статистической обработкой полученных данных. В основном исследование ВС с помощью имитационных моделей является статистическим моделированием.

Наиболее важными исследуемыми характеристиками ВС являются показатели эффективности, под которыми понимается общая производительность, выполнение обслуживания отдельных заявок в единицу времени и т. д. Находя значения показателей эффективности, можно оценить эффективность различ-

ных способов управления, получить оценки вариантов структуры ВС, влияние параметров компонентов системы на ее функционирование и т. д.

Суть имитационного моделирования ВС заключается в рассмотрении функционирования ее отдельных составных частей. Система (аппаратная или программная) разбивается на компоненты так, чтобы можно было просто описать поведение каждого из них. Затем описывается взаимодействие между компонентами. Если правила поведения частей системы и их связи достаточно хорошо описывают реальное поведение, то полученная модель может служить представлением реальной системы. Подобные модели разрабатываются с использованием специальных языков, имеют сложную структуру и требуют больших затрат памяти и времени реализации.

Имитационная модель ВС. Рассмотрим составные части имитационных моделей. Модель источника входного потока представляет собой алгоритм, по которому вычисляются моменты появления заявок. При этом источники могут быть зависимыми и независимыми. Модель *независимого источника* чаще реализует алгоритм выборки значений случайной величины, распределенной по заданному закону. В *зависимых источниках* заявка на выходе вырабатывается при поступлении на входе другой заявки.

Ресурсы ВС делятся на устройство и память. Устройство может обслуживать в момент времени одну заявку, а память – несколько. Модель устройства есть алгоритм выработки значений интервалов обслуживания. В модели для каждого типа заявок могут быть установлены свой закон распределения и его параметры. Модель устройства отражает также заданную дисциплину обслуживания путем управления очередями и приоритетами поступивших заявок.

Модель памяти включает алгоритм для вычисления объема, требующегося для обслуживания заявки. Объем памяти определяется как реализация случайной величины, причем закон распределения зависит от типа заявки. Характеристики памяти включают ее емкость и дисциплину обслуживания. Заявка, поступившая в память, занимает объем и движется до встречи с элементом, вызывающим освобождение памяти.

Связь источников и ресурсов определяется программными средствами системы. Для имитации связи в модели используются специальные операторы, называемые *узлами*. Узлы первого типа служат для направления заявок по тому или иному маршруту в зависимости от выполнения условий. Узлы второго типа разделяют заявки или их объединяют, третьего типа предназначены для изменения параметров заявок.

Таким образом, имитационная модель ВС представляет собой алгоритм, состоящий из описания моделей элементов – источников, устройств, памяти, узлов и обращений к ним. В процессе имитации в модели происходит изменение дискретного времени, после имитации – очередной группы событий, относящихся к текущему времени t_1. Интервал t_i увеличивается на Δt, отделяет данную группу событий от последующих. В процессе имитации происходит накопление данных: количества заявок, вышедших из системы обслуженными и необслуженными, прошедшие обслуживание в каждом из устройств, время состояния занятости каждого устройства, средние значения используемой памяти, длин очередей и т. д. Процесс имитации заканчивается по истечении времени моделирования или выработки определенного количества заявок. После этого происходит обработка накопленных данных в процессе имитации.

Языки имитационных моделей ВС. Рассмотрим особенности языков процедурного типа на примере языка GPSS. Первую группу операторов составляют операторы, позволяющие отображать действие заявок по отношению к ресурсам (приход и уход заявок из устройств, вхождение в память и ее освобождение, захват занятого устройства). Во вторую группу входят операторы для управления движением заявок в модели ВС. Третью группу составляют операторы, предназначенные для выполнения арифметических операций и других процедур, подсчета и вывода результатов моделирования и т. д.

Операторы языка состоят из трех частей: метки, операции и переменных. Первые два поля аналогичны другим языкам. Поле переменных состоит из подполей, обозначаемых A, B, C, D, E, F, G, H , отделяемых друг от друга запятыми. Назначение подполя задается при описании оператора.

Для генерации заявок используется оператор GENERATE со следующими значениями подполей: A – среднее время между появлениями заявок t_2; B – среднее время отклонения от t_2; C – интервал времени до появления первой заявки: D – количество заявок; E – приоритет заявок. Значение оператора GENERATE 2 означает, что заявки будут поступать через две единицы времени. Для остальных подпараметров B = 0, C = A, D = ∞, E = 0 по умолчанию.

Оператор ASSIGN служит для задания значения параметру заявки, причем задаваемое значение в подполе B присваивается значению A.

Операторы QUEUE A и DEPART A обозначают соответственно вхождение заявки в очередь A и выход из нее. В операторах SEIZE и RELEAZE в подполе A указывается имя обслуживаемого ресурса при его занятии и освобождении.

Оператор ADVANCE A, B используется для задержки заявки при обслуживании и имеет такое же назначение подполей, как оператор GENERATE.

Для размножения транзактов используется оператор SPLIT . Основной транзакт направляется к следующему оператору, а копии, количество которых определяется в подполе A, поступают к оператору, метка которого задается в поле B.

Оператор START означает конец ввода. В подполе A указывается число, засылаемое в итоговый счетчик. Этот оператор вызывает вывод на печать всех накопленных статистических данных по ресурсам, очередям и транзактам. Шаг вывода на печать указывается в подполе C. Блокировка печати осуществляется по символу NP в подполе B.

Оператор TERMINATE служит для уничтожения транзактов. В подполе указывается число, вычитаемое из итогового счетчика. Счет завершается при нулевом значении этого счетчика.

Оператор TRANSFER является оператором перехода и имеет несколько разновидностей, задаваемых в подполе A. Если A пусто, то осуществляется переход к оператору, указанному в подполе B. Если в A указано число в виде правильной дроби, то оно приравнивается вероятности q , с которой осуществляется переход к оператору, метка которого указана в C. Переход к оператору, указанному в B, произойдет с вероятностью $1 - q$. Если в подполе A указан символ P, то номер оператора, к которому происходит переход, образуется сложением значения в подполе C со значением параметра транзакта, номер которого указывается в B.

Для задания функции служит оператор FUNCTION , у которого в поле

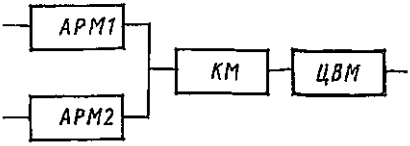


Рис. 9.3.
Структура моделируемой ВС.

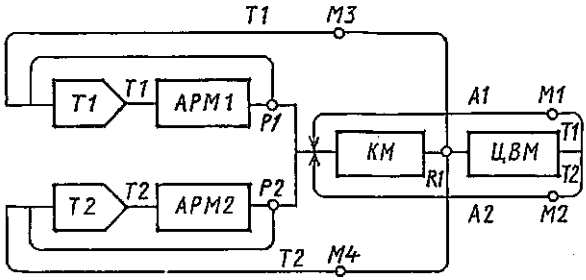


Рис. 9.4.
Схема сетевой имитационной модели для ВС, представленной на рис. 9.3.

метки записывается указатель функции, в подполе А — тип аргумента, в подполе В — число пар координат, далее значения координат. Если аргумент — равномерно распределенная величина в интервале [0, 1] , то в подполе А записывается символ RM .

Более подробно язык GPSS описан в [29].

Пример. Описать на языке имитационного моделирования ВС (рис. 9.3 , на котором КМ – коммутатор, АРМ – автоматизированное рабочее место).Нужно оценить загрузку КМ, ЦВМ и среднее время ожидания обслуживания заявок, поступающих из АРМов. Первая стадия заключается в составлении сетевой имитационной модели (рис. 9.4), в которую дополнительно вводятся источники заявок Т1–Т2 и узлы трех типов. Каждый из узлов Р1, Р2 направляет заявку на один из двух выходов в соответствии с вероятностями. Узел R1 распределяет заявки по возможным направлениям. Узлы М1–М4 служат для изменения типов заявок. Т1, Т2 имитируют работу оператора АРМа за экраном дисплея. Работа начинается с генерирования заявок по одной, которые обрабатываются в мини-ЭВМ и в зависимости от характера решаемых задач могут вызвать запрос к оператору или обращение к ЦВМ. Заявки, направленные в ЦВМ, проходят через КМ (возможно, с ожиданием в очереди), а затем через R1 поступают в ЦВМ, где либо обслуживаются, либо ожидают в очереди. Затем через коммутатор они направляются в АРМ, при этом для отличия их от Т1, Т2 обслуженные заявки преобразуются в узлах М1, М2 (в типы А1, А2 соответственно). После направления заявок в нужные АРМы в узлах М3, М4 они восстанавливаются. В АРМе поступившие данные могут выводиться на печать, обрабатываться либо инициировать генерацию новых заявок. Далее составляется программа. Рассмотрим ее фрагмент для моделирования АРМ 1.

```
1.        GENERATE ,,, 1          Создается один транзакт
2.  M1    ASSIGN 1,M1             Транзакту присваивается имя М1
3.        SEIZE APM1              Транзакт входит на обслуживание в АРМ
4.        ADVANCE 2·F2            Определяется время обслуживания 2·F2
5.        RELEAZE APM1            Транзакт завершает обслуживание в АРМ
```

6.		TRANSFER 0.75,M7,M2	С вероятностью 0,75 транзакт возвращается в дисплей Т1 и с вероятностью 0,25 – в КМ (метка М7)
7.	M2	SEIZE T1	Транзакт снова входит в АРМ
8.		ADVANCE 4,F2	Происходит его задержка на 4·F2, имитирующая работу оператора
9.		RELEAZE T1	Освобождается устройство
10.		TRANSFER .4,M1,M11	С вероятностью 0,4 задача будет решена и переходит к обработке в ЦВМ (М11)
11–20.		. . .	Аналогично описывается второй АРМ
21.	M7	ASSIGN 2,M8	Второму параметру присваивается значение М8=28
22.	M9	QUEUE K1	Постановка транзактов в очередь К1
23.		SEIZE KM	Поступление транзактов на обслуживание в КМ
24.		DEPART K1	Выход из очереди К1
25.		ADVANCE 1,F2	Задержка в коммутаторе на величину F2
26.		RELEAZE KM	Освобождение коммутатора КМ
27.		TRANSFER P,2,0	Транзакты с параметром М8 поступают в К2,М10–в 35
28.	M8	QUEUE K2	Постановка транзакта в очередь К2
29.		SEIZE ЦВМ	Транзакт входит на обслуживание в ЦВМ
30.		DEPART K2	Освобождение очереди К2
31.		ADVANCE 12,F2	Задержка на обслуживание в ЦВМ 12·F2
32.		RELEAZE ЦВМ	Освобождение ЦВМ
33.		ASSIGN 2,M10	Изменение второго параметра транзакта М10=35
34.		TRANSFER M9	Передача транзактов снова в очередь К1
35.	M10	TRANSFER P,1,1	Передача в свой АРМ для продолжения
36.	M11	SPLIT 1,M10	Транзакт передается на оператор 37, копия –на М10
37.		TERMINATE 1	Вычитание единицы из счетчика
38.		START 100	Конец моделирования, в счетчик первоначально занесено число 100
39.		END	

9.4. МОДЕЛИРОВАНИЕ ПРОЦЕССОВ НА ОСНОВЕ СЕТЕЙ ПЕТРИ

Основные определения. При проектировании и анализе ВС и программного обеспечения в последние годы одним из методов моделирования является использование сетей Петри. Сети Петри, оперирующие понятиями "события" и "условия", широко применяются для моделирования в основном параллельных процессов. Они, как правило, ориентированы на получение качественных оценок исследуемой системы, для исследования логической корректности алгоритмов функционирования ВС, конечности множества ее состояний, достижимости некоторого состояния и т. д.

Сеть Петри задается совокупностью множеств $N = (P, T, A, B, \vec{M}_0)$, где P – множество позиций $|P| = m$; T – множество переходов $|T| = n$ и $P \cap T = \varnothing$; A – функция входных инциденций $(A: P \times T \to N, \quad N = 1, 2, ...)$; B – функция выходных инциденций $(B: P \times T \to N)$; $\vec{M}_0$ – вектор начального маркирования $(|\vec{M}_p| = m)$.

Функции входных и выходных инциденций задают отношения между позициями и переходами сети. Множество ${}^0t$ (t^0) называется множеством входных (выходных) позиций перехода t, а множество ${}^0p(p^0)$ – множеством

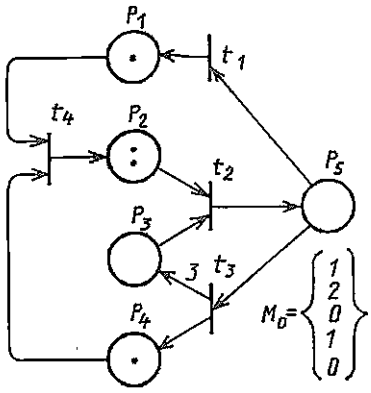


Рис. 9.5.
Графическое представление сети Петри.

входных (выходных) переходов позиции p. Вектор $\vec{M}_0$ ставит в соответствие каждой позиции сети целое неотрицательное число.

Графически сеть Петри представляет собой ориентированный двудольный граф (рис. 9.5), вершинами которого являются позиции (кружки) и переходы (черточки). Ребра графа определяются функциями входных и выходных инциденций. Если $A(p_i t_j) = a_{ij} \neq 0$, то существует ребро от позиции p_i к переходу t_j с весом a_{ij}. Если $B(p_i t_j) = b_{ij} \neq 0$, то существует ребро от перехода t_j к позиции p_i с весом b_{ij}. Если вес ребра равен 1, оно не помечается. Сеть Петри, у которой все ребра равны 1, называется *ординарной*. В содержательном плане позициям соответствуют условия, а переходам – события. Входные позиции перехода t определяют набор условий, необходимых для того, чтобы событие (t) могло произойти. В результате события появляются новые условия, определяемые выходными позициями перехода t.

Наличие того или иного условия интерпретируется с помощью меток маркеров, изображаемых в виде точек внутри позиции. Текущее состояние процесса полностью определяется распределением маркеров по позициям сети и может быть задано в виде вектора маркирования $\vec{M} = \{M(p_1), ..., M(p_m)\}$, где m – число позиций, а $M(p_i)$ равно числу маркеров внутри позиции p_i. Исходному состоянию системы соответствует вектор начального маркирования $\vec{M}_0$ (см. рис. 9.5).

Динамика изменения состояний моделируется перемещением маркеров по позициям сети в результате срабатывания возбужденных переходов. Переход t называется *возбужденным*, если в каждой входной позиции количество маркеров больше или равно весу ребра. При срабатывании перехода t из всех его входных позиций p_i изымается $A(p_i, t)$ маркеров, а в каждую исходную позицию p_j добавляется $B(p_j t)$ маркеров. Срабатывание любого перехода t_i при начальном маркировании $\vec{M}_0$ приводит к новому маркированию $M_1 = \{0, 4, 0, 0, 0\}$ (см. рис. 9.5). Изменение маркирования может продолжаться долго, пока есть хотя бы один возбужденный переход.

Применение сетей Петри для описания систем. Представление реальной системы в виде сети Петри есть формальная и наглядная запись алгоритма функционирования. Сеть может рассматриваться как язык описания и общения. Во-первых, все остальные применения базируются на интерпретации процесса сетью Петри, во-вторых, можно описывать как программные, так и аппаратные средства, в-третьих, сам процесс конструирования сети порождает вопросы о действии реальной системы.

Основным достоинством сетей Петри как языка описания является наглядное отображение параллелизма. Переходы сети, составляющие шаг, мо-

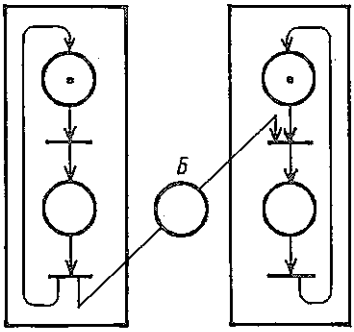


Рис. 9.6.
Синхронизация процессов через буфер.

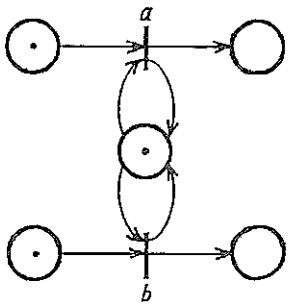


Рис. 9.7.
Моделирование разделения ресурсов.

гут срабатывать независимо, т. е. события будут происходить параллельно и асинхронно. В то же время сетью могут описываться механизмы синхронизации, например отражать синхронизацию процессов, взаимодействующих через буферный накопитель. Процесс чтения может начаться только при заполнении буфера Б (рис. 9.6). Ситуации, свойственные конкурированию процессов за ресурсы, также описываются сетями Петри (рис. 9.7). Переходы *a* и *b* находятся в конфликте: несмотря на то что возбуждены оба перехода, сработать (войти в критический участок) может только один из них. Хотя в явном виде сеть Петри не содержит временных характеристик процессов, она отражает временную последовательность событий. В сети присутствуют отношения типа раньше–позже, одновременно, независимо и т. д. Сеть Петри может моделировать различные уровни детализации описания ВС.

Моделирование элементов параллелизма. Параллелизм в сетях Петри может быть введен несколькими способами. Рассмотрим случай двух параллельных процессов, каждый из которых может быть представлен сетями Петри. В этом случае составная сеть является объединением сетей для каждого из двух процессов. Начальная маркировка составной сети имеет две метки по одной в каждой сети, представляя начальный счетчик команд процесса.

Одним из подходов является моделирование операций FORK и JOIN (рис. 9.8, *а, б*). Выполняющийся процесс *i* создает два новых *j* и *k* оператором FORK, JOIN соединяет два процесса, которые сливаются в один процесс, продолжающийся на участке К.

Другое предложение по введению параллелизма основано на операциях PAR и END . Структура управления имеет вид PAR S_1 , S_2 , ..., S_n END, где S_i – предложение. Смысл структуры заключается в параллельном выполнении каждого из предложений S_i . Эта конструкция может быть представлена сетью Петри (рис. 9.8, *в*).

Для взаимодействия параллельных процессов требуется разделение ресурсов между ними. Проблемы синхронизации, возникающие при взаимодействии процессов, иллюстрируются многочисленными примерами, среди которых задача о взаимном исключении, производителе-потребителе, чтении-записи и т. д. На рис. 9.6 и 9.7 показано моделирование сетями Петри двух из вышепе-

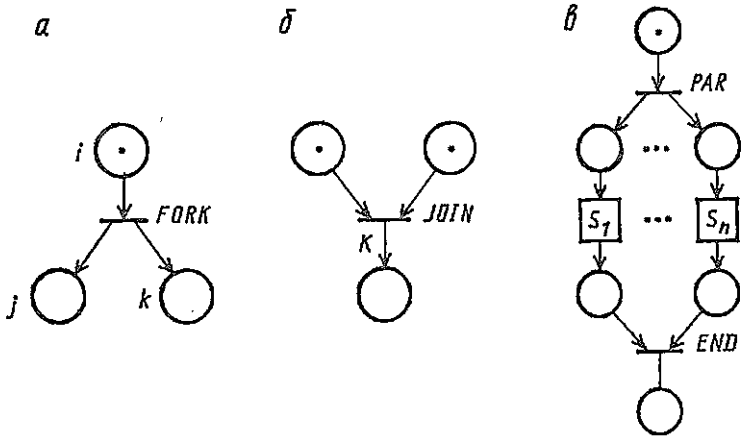


Рис. 9.8.
Моделирование элементов параллелизма.

речисленных задач. Один из классических способов синхронизации P- и V-операции над семафорами также легко моделируется сетями Петри (рис. 9.9). Каждый семафор моделируется позицией, количество меток в которой показывает значение семафора. Операция P использует позицию семафора в качестве входа, а V – в качестве выхода.

Сети Петри для имитационного моделирования. При моделировании систем сетями Петри маркеры, как правило, соответствуют объектам или ресурсам системы. Эти ресурсы могут иметь свойства, которые трудно интерпретировать обычными маркерами. Например, на рис. 9.10 показана сеть Петри, моделирующая часть ОС, обслуживающую распределенную систему дисковой памяти (рис. 9.11). Маркеры в позиции p_1 соответствуют свободным каналам, а в позиции p_2 – незанятым дисководам. Хотя каналы и дисководы идентичны, их взаимодействие в системе требует, чтобы дисковод Д1 работал только с каналом А, дисковод Д3 – с каналом В, а Д2 может использовать как канал А, так и В. Эти ограничения усложняют моделирование системы обычной сетью Петри. Для облегчения моделирования предлагается применять сети с цветными маркерами. Ресурсам или объектам со специфическими свойствами соответствуют маркеры разных цветов, а для каждого перехода вводится функция, определяющая маркирование выходной позиции в зависимости от цветов маркеров.

Для моделирования процессов в ВС были разработаны оценивающие сети (*E*-сети), достоинством которых является получение временны́х характеристик процессов и непосредственная интерпретация системы как имитационной модели. *E*-сеть определяется как связная конфигурация позиций посредством разрешенных схем переходов и задается пятеркой $E = (L, P, R, A, \vec{M}_0)$, где L – множество позиций; P – множество периферийных позиций; R – множество решающих позиций; A – множество характеристик перехода $\{a_i\}$, $a_i = (s, t(a_i), q)$, s – схема перехода; $t(a_i)$ – время перехода; q – процедура перехода; $\vec{M}_0$ – начальное маркирование. Маркеры в *E*-сети могут быть прос-

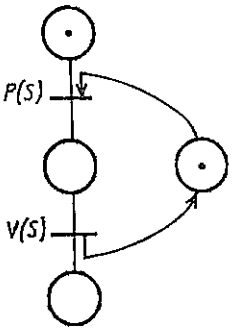


Рис. 9.9. Моделирование P и V операций над семафором S

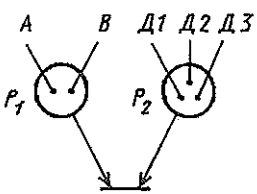


Рис. 9.10. Сеть Петри, моделирующая часть ОС.

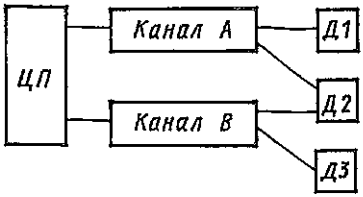


Рис. 9.11. Структура части ВС.

тыми и сложными, представляющими совокупность атрибутов, которые изменяются при срабатывании перехода.

E-сети используются для моделирования ВС на уровне задача–ресурс. Они позволяют проверить общую работоспособность системы и проследить за прохождением потоков управления и данных. При этом множество внешних переменных может носить и случайный характер, т. е. *E*-сеть может быть как детерминированной, так и вероятностной.

9.5. ПРОЕКТИРОВАНИЕ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ В ХОДЕ МОДЕЛИРОВАНИЯ

Последовательность проектирования. Проектирование вычислительных процессов в ВС производится на основе аналитического описания, составления имитационной модели, а также комбинированным способом. В процессе проектирования можно выделить следующие стадии: содержательное описание разрабатываемой ВС (постановка задачи); составление формального описания ВС и преобразование его в модель; программирование модели ВС, ее испытания; принятие решений по результатам моделирования. В качестве средств автоматизации проектирования процессов рассматриваются системы моделирования и оценки результатов. В качестве примера рассмотрим упрощенный ход моделирования многопроцессорной ВС.

Содержательное описание. Пусть необходимо спроектировать ВС, включающую несколько процессоров и работающую в трех режимах: пакетной обработки, диалоговом и смешанном. В состав ВС должны входить следующие блоки:

1) обрабатывающие процессоры, выполняющие только вычислительные работы и прерывающиеся при вводе-выводе, отказе, программной ошибке;

2) управляющий процессор (монитор), реализующий алгоритмы ОС и поддерживающий механизмы прерывания;

3) система прерываний, обрабатывающая сигналы прерывания трех уровней с возможностью маскирования причин прерывания;

4) оперативная память ($ОП_i$), состоящая из блоков, в каждом из которых помещается одна рабочая программа, выполняемая на одном процессоре;

5) коммутатор памяти для обеспечения доступа процессора и периферийных устройств к каждой из секций ОП;

6) контроллер внешней памяти (КВП) для обмена между внешней и оперативной памятью;

7) селектор каналов (СК), управляющий работой периферийных устройств, включая терминалы.

Программное обеспечение ВС содержит ОС и системные программы. ОС выполняет три основные функции: циклический осмотр состояния системы, обработку прерываний и обработку системных команд. При этом организуется сохранение информации ОС, прерванной по различным причинам. Появление запроса прерывания приводит к передаче управления блоку обработки прерываний монитора, после обработки прерывания возобновляется работа ОС.

По запросам пользователей ОС создает диалоговые процессы, которые вызывают выполнение программ из $ОП_i$ на обрабатывающих процессорах. Для этого в состав ОС входит блок процессов, который организует очереди процессов, производит их взаимодействие, смену процессов в $ОП_i$.

Системные программы выполняют диалоговое взаимодействие, ввод информации, пакетную обработку. Одновременно несколько программ могут нуждаться в обработке или находиться на различных ее стадиях. Каждый раз при освобождении процессора монитор подключает его к блоку $ОП_i$, при этом может организовываться вывод программы на внешнюю память.

Составление формального описания ВС. На этапах проектирования ВС возникают задачи выбора состава и структуры системы. В качестве примера рассмотрим выбор архитектуры из трех вариантов: симметричная организация управления и обработки (процессоры и монитор – однотипные устройства); асимметрия на обработку (управление) – скорость монитора ниже или выше скорости процессоров. Варианты будем оценивать на основе следующих критериев эффективности: средней стоимости ожидания ответов абонентами P_1 и приведенного коэффициента загрузки процессоров P_2.

Для вычисления показателей зададимся стоимостью оплаты времени пользователей C_{a_1} , вектором коэффициентов k_j , компоненты которого определяют значимость ожидания ответа j абоненту: коэффициентами значимостей загрузки процессора a_1 и монитора a_2 . Тогда качество вариантов ВС можно оценить по формулам:

$$P_1 = (C_{a_1}/4T_м)\sum_{j=1}^{d_0} k_j \sum_{L=1}^{4} T_{1j}\,;\quad P_2 = (a_1/T_м)\sum_{k=1}^{s_0} t_{pk} + (a_2/T_м)\,t_{pм}\ ,$$

где $T_м$ – длительность моделирования; T_{ij} – среднее время ответа на i запрос; t_{pk} , $t_{pм}$ – общее время работы k-го процессора и монитора за время $T_м$; d_0 – число абонентов; s_0 – число модулей монитора. Имеются 4 вида запросов абонента: реакция на обработку начала сеанса абонента, реакция на ввод-вывод, время ожидания ответа ВС, время реакции на сигнал пользователя после окончания вывода информации. В ходе моделирования необходимо опреде-

лить набор трех типов статистик T_{ij}, t_{pk}, t_{pm}. Параметры d_0 и s_0 заданы.

Для отладки модели и анализа вариантов архитектуры необходимо фиксировать следующие статистики: среднее время ввода информации в ВС, количество запусков пакетов; среднее количество введенных заданий в пакете; средняя длительность сеанса взаимодействия абонента с ВС; количество взаимодействий за сеанс и т. д.

Для моделирования необходимо задать ряд значений, определяющих состав и структуру внешней среды и характеристики используемого оборудования, таких, как среднее время подготовки абонента к сеансу, обдумывания решения; средние значения времени обменов $ОП_i$ с лентами и дисками; число различных типов прерываний и т. д. Все задаваемые параметры формируются на основе результатов измерений системы прототипа. Делается допущение о стационарности процессов в системе, а различные частотные параметры используются для вычисления соответствующих вероятностей.

Описание алгоритмов. Исходя из функции моделирования, задач проектирования и состава параметров моделирования определяется уровень детализации процессов. Ресурсное представление взаимодействия компонентов для задачи моделирования несущественно, поэтому допускается, что размеров внешней памяти достаточно для организации процессов. Как несущественная не рассматривается модель коммутатора $ОП_i$ с процессорами СК и КВП. Алгоритм функционирования КВП распадается на ряд алгоритмов обменов с МЛ и МД.

Модель системы прерываний представляет набор очередей, в которые поступают требования на обслуживание монитором трех типов прерываний. Работу процессора и таймера можно объединить в одном алгоритме функционирования.

Аналогично упрощаются алгоритмы представления модулей ОС: можно исключить из рассмотрения работу с ресурсами, массивами. В полном объеме представлены блоки каналов, физического обмена и работы с процессорами. Связь монитора с СВМ, СК и процессором отражается системой очередей.

Следующим шагом формализации является описание информационных связей между элементами ВС в виде модели массива обмена абонентов с системой. Дальнейшая формализация ВС состоит в рассмотрении временно́й диаграммы функционирования каждого компонента, включая и программное обеспечение. На этой основе составляется общий алгоритм функционирования. Затем детально проводится формальное описание функционирования оборудования и операционной системы. При этом отдельно составляются алгоритмы для каждого процесса ОС обслуживания прерываний.

Преобразования формального описания в модель. С этой целью необходимо каждому процессу, кроме его алгоритма, обеспечить внутреннюю и внешнюю синхронизацию. Внутренняя синхронизация взаимодействия процессов обеспечивается с помощью семафоров. Внешняя синхронизация выполняется системой моделирования. Для этого используются операторы синхронизации процессов типа WAIT. Поскольку алгоритмы процессов из-за значительных упрощений объекта оказались несложными, удалось избежать деления их на части.

Информационную базу модели составляют параметры моделирования: длительность задержки процессов, частота появления событий в процессах,

функция распределения длительности работы процессов, которые собираются в виде единого массива. Основная часть имитационной модели представляет описание процессов, внутри которых установлены операторы синхронизации в модельном времени.

Следующим шагом преобразования формального описания в модель является выбор мест установки операторов сбора статистики. Исходя из функции моделирования, определяем следующие типы статистик: коэффициент использования процессов N_i, характеристики использования очередей Q_i, трассировки обслуживания требований процессами *NP*. Операторы сбора статистик N_i помещаются за операторами синхронизации, а Q_i – в местах записи и чтения запросов из очередей. При каждом инициировании процессов накапливаются значения счетчиков числа их запусков. За ходом имитации вводится процесс для контроля.

Процесс моделирования. При выборе языка программирования для модели необходимо рассмотреть такие особенности, как процессорный подход к реализации компонентов модели, наличие процедур испытаний, простоту задания множества однотипных процессов, наличие стандартных средств фиксации статистики, возможность установки трасс, наличие средств обработки эксперимента. Этим требованиям удовлетворяют языки PLSIM и GPSS. После описания модели начинается фаза автономной отладки процессов, что требует задания каждому процессу имитатора внешнего окружения. В фазе комплексной отладки структура модели усложняется постепенно: наращивается цепочка взаимодействующих процессов. Необходимо хорошо организовать дампирование процессов. В результате отладки получается имитационная модель, которая нуждается в испытании.

Для проверки адекватности модели реальной системе-прототипу был выбран вариант структуры ВС, включающий процессор, несколько абонентов, печать, перфокарточный ввод. Испытания проводились при соотношении скоростей монитора и процессора $K_0 = 4$. Показатели качества модели и реального объекта различались после $T_м = 3000$ с времени эксперимента не более чем на 10 %.

После установки адекватности модели реальной системе проводятся эксперименты по исследованию влияния архитектуры ВС на организацию вычислительных процессов.

Вначале определялась рабочая длина реализации модели, для чего было просчитано семь вариантов с одними и теми же параметрами управления моделью, но с разными значениями времени моделирования, при этом получены зависимости P_1 и P_2 от величины $T_м$. На основании этого определено, что установившийся процесс имеет место уже при $T_м = 3000$ с, когда оба показателя качества P_1 и P_2 стабилизируются.

В ходе имитационного эксперимента длительностью 3000 с было проведено 22 сеанса работы. За один сеанс пользователь в среднем 1,6 раза взаимодействовал с ОС. Модельный эксперимент дал следующие результаты: $P_1 = 6{,}7$ руб/ч, $P_2 = 0{,}85$ руб/ч. Это не более 10 % расхождения с реальной системой, что достаточно для выводов о выборе архитектуры ВС.

Результаты моделирования ВС. Основная задача исследования на модели – оценка влияния архитектуры ВС на организацию вычислительного процесса. Были исследованы три варианта архитектуры: симметричная и две асиммет-

Таблица 9.1

Симметричная структура			Асимметрия на обработку			Асимметрия на управление		
K_0	P_1	P_2	K_0	P_1	P_2	K_0	P_1	P_2
1,0	11,6	0,593	1,5	0,192	0,7	1,5	0,216	0,46
4,0	3,45	0,589	2,0	1,5	0,72	2,0	1,88	0,4
16,0	1,0	0,578	3,0	3,92	0,76	3,0	4,42	0,37
41,0	0,5	0,58	3,5	5,28	0,841	3,5	6,87	0,281
100,0	0,144	0,588	4,0	6,7	0,85	4,0	9,6	0,266

ричных. В первом случае монитор и ЦП имели одинаковое быстродействие. Во втором случае процессор обладал бо́льшим быстродействием, чем монитор, в третьем — меньшим, чем монитор. Задача сформулирована так: для каждого варианта архитектуры определить уравнение $P_i = f(K_0)$, где K_0 – относительная производительность процессоров. Это уравнение строится по табличным значениям, полученным в результате прогона модели (табл. 9.1).

Поведение многопроцессорной ВС рассматривалось с двух сторон: организация процессов и обслуживания абонентов. Анализ показал, что симметричная архитектура недогружена, поэтому коэффициент загрузки P_2 с увеличением скорости обработки информации K_0 практически не изменяется. На обслуживание абонентов большое влияние оказывает скорость обработки информации в ЦП. С увеличением скорости обработки информации K_0 коэффициент загрузки системы P_2 увеличивается, что приводит к снижению времени ожидания абонента. Но из-за недостаточной скорости работы монитора возрастает удельная стоимость ожидания абонентов, что приводит к возрастанию оценки P_1.

Увеличение степени асимметрии на обслуживание снижает загрузку процессоров P_2, т. е. процессор не успевает справиться с заказами абонентов на обработку, что ведет к росту P_1.

Таким образом, можно сделать вывод о преимуществе симметричной архитектуры по сравнению с остальными, особенно если в качестве процессоров обработки и монитора использовать ЭВМ ЕС 1035, ЕС 1036 и др. Однако эти выводы справедливы при заданном количестве абонентов. При их увеличении недогрузка системы будет падать.

Выводы

1. Автоматизация проектирования вычислительных процессов в ВС как на уровне аппаратных средств, так и программных, требует разработки адекватных моделей процессов и средств их реализации на ЭВМ. В качестве моделей процессов используются аналитические, имитационные и модели на базе сетей Петри. Системы моделирования процессов на ЭВМ строятся на базе различных языков: организации взаимодействия транзактов, управления процессами, составления расписаний событий.

2. Аналитические модели, служащие для предварительных оценок характеристик ВС, могут базироваться на методах теории СМО. Функционирование СМО определяется совокупностью трех характеристик: входного потока заявок, параметров обслуживающих ресурсов, дисциплин обслуживания заявок ресурсами. Наиболее простой моделью вычис-

лительного процесса является марковская, оценивающая трудоемкость алгоритмов.

3. Имитационные модели базируются на программной имитации процессов функционирования ВС для определения одной или нескольких основных характеристик системы и предназначены для количественной оценки результатов, полученных аналитически.

4. Для имитации процессов ВС, особенно параллельных, используются сети Петри, которые позволяют получить в основном качественные характеристики: корректность алгоритмов функционирования ВС или ее части, конечность множества ее состояний, отсутствие конфликтов и т. д.

5. Процесс автоматизации проектирования вычислительных процессов может включать: содержательное описание ВС (ее части); формальное описание работы ВС и оценок ее функционирования; преобразование формального описания в модель; программирование этой модели и ее испытание; оценки результатов моделирования и внесение возможных изменений.

Вопросы, изложенные в § 9.1–9.3, рассматриваются в [1, 12, 14, 17, 29], в § 9.4 – [18], в § 9.5 – [12, 14, 17, 29].

Вопросы для контроля

1. Какие методы используются для описания функционирования ВС?
2. Поясните основные понятия системы массового обслуживания.
3. Охарактеризуйте модели программных компонентов ВС.
4. Какие потоки используются для описания поступления заявок?
5. Поясните виды дисциплин обслуживания заявок.
6. Как моделируется длительность обслуживания заявок?
7. Поясните принцип имитационного моделирования компонентов ВС.
8. Объясните особенности языка GPSS.
9. Как моделируется сетью Петри синхронизация процессов?
10. Как моделируется сетью Петри распределение ресурсов?
11. Опишите последовательность проектирования вычислительного процесса.

10. ВЫЧИСЛИТЕЛЬНЫЕ ПРОЦЕССЫ И ЭВМ ПЯТОГО ПОКОЛЕНИЯ

10.1. КОНЦЕПЦИИ И ОРГАНИЗАЦИЯ ЭВМ ПЯТОГО ПОКОЛЕНИЯ

Предпосылки разработки. Одной из причин разработки ЭВМ следующих поколений является несовершенство архитектуры большинства современных ВС. Это заключается в сравнительно небольших структурных изменениях в аппаратурных решениях, практически не учитывающих усложнения языков и ПО. Такая ситуация привела в середине 70-х годов к семантическому разрыву между архитектурой ЭВМ и языками программирования, организацией памяти, операционными системами. Последствиями этого разрыва являются ненадежность программного обеспечения, невысокая эффективность ВС в связи с примитивной системой команд, из-за которой компилятор генерирует, а машина интерпретирует большое их количество. Как следствие этого – сложность компиляторов, невысокая продуктивность программирования, трудности отладки программ [11].

Для совершенствования архитектуры ВС используются новые принципы организации памяти с введением самоопределяемых данных (теговая память), введение областей памяти с санкционированным доступом, аппаратурная реализация функций ОС (управление процессами, распределение ресурсов), использование кэш-памяти. В области обработки однопроцессорные структуры заменяются многопроцессорными и многомашинными.

В области программирования наметился переход от составления алгоритмов решения задачи к описанию постановки задачи. Языки программирования заменяются языками спецификаций. Начинают широко использоваться языки проектирования систем АДА, ОККАМ и язык логических выводов ПРОЛОГ. В области данных наметился переход от их обработки к обработке элементов знаний.

Концепции. Проводимые в течение последних лет исследования в таких областях, как экспертные системы, интеллектуальные интерфейсы и нетрадиционные архитектуры ЭВМ, позволили определить основные характеристики машин пятого поколения. Среди них можно выделить следующие: значительное расширение выполняемых функций, типов и уровней ЭВМ, уменьшение универсальности и возрастание роли специализированных ЭВМ, развитие архитектурных решений, отличных от архитектуры фон Неймана.

Для обеспечения взаимодействия с человеком существенно возрастает роль диалога, причем в различных формах: текстовых, графических, с голоса и т. д. При этом система должна быть способна вести интеллектуальный диалог: отвечать на вопросы в произвольной форме, выдвигать предложения, давать подсказки, производить обобщающие выводы и т. д. Решающее значение в этом направлении имеет использование принципов решателей задач (программных средств поиска решений в пространстве состояний), которые широко применялись для узких областей: в шахматах, математике и т. д.

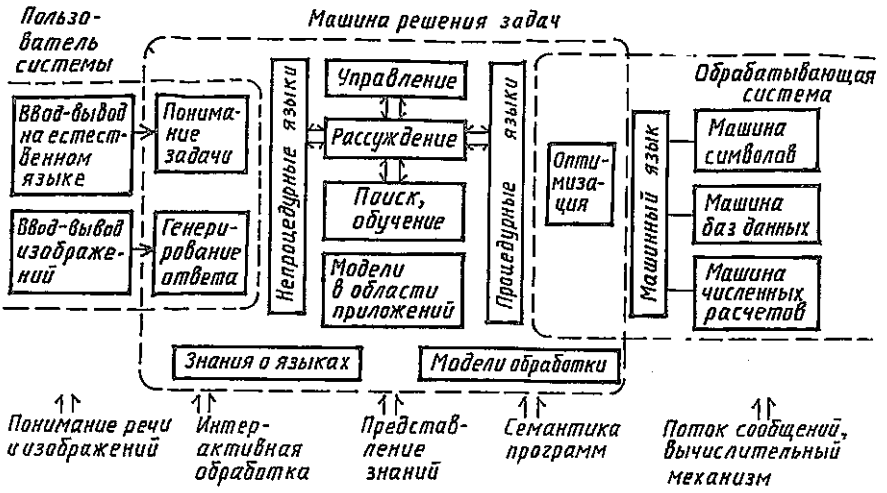


Рис. 10.1.
Система представления знаний и технология ее реализации.

Еще одной важной функцией является приобретение новых знаний, связанных с конкретными предметными областями. Все это привело к разработке архитектуры, реализующей описанные выше функции (рис. 10.1). Для осуществления ряда функций необходима обработка не числовых, а символьных данных, а для реализации процессов обучения и рассуждения – недетерминированные алгоритмы. Большое внимание уделяется вопросам систематического программирования, основанного на абстрагировании данных и методах преобразования программ. Чтобы обеспечить способность к рассуждению, необходима разработка машины реляционной алгебры для выполнения поиска в реляционной базе данных.

Основные функции таких ЭВМ можно представить следующим образом: решение логических задач и получение выводов; создание и управление базой знаний; обеспечение интеллектуальных интерфейсов (на естественном человеческом языке с экрана и голоса, на основе графических образов); автоматизация программирования решаемых задач.

Схема, представленная на рис. 10.2, иллюстрирует представление о взаимодействии аппаратурных и программных средств при реализации данных функций. Главный объект программных разработок представлен в виде моделирующей программной системы, а аппаратурных разработок – в виде системы машины. Верхняя часть моделирующей программы отражает функции, связанные с решением задач и получением выводов, нижняя – средства управления базой знаний. Любая система решения задач может использовать большие внешние базы данных, функционирующие как библиотеки.

Основной отличительной чертой аппаратурных средств является поддержка решения задач (аппаратурная реализация механизмов поиска в пространстве состояний), работы с базами знаний (аппаратурная поддержка функций системы управления базами знаний). Это в свою очередь требует аппаратурной

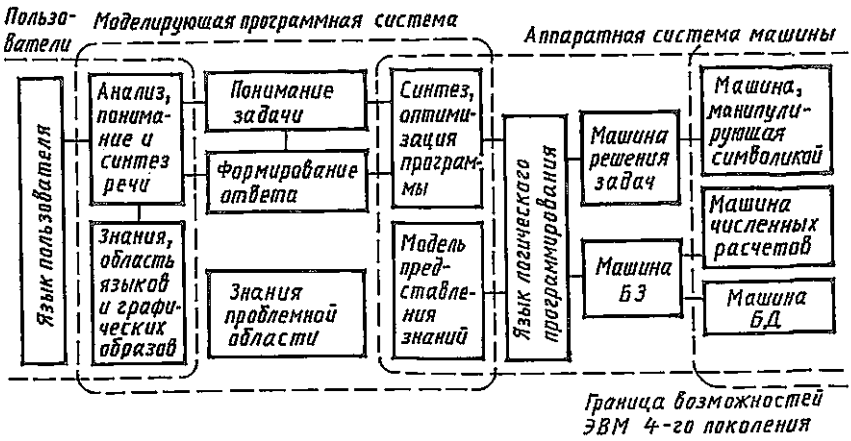


Рис. 10.2.
Концептуальное представление об ЭВМ пятого поколения.

поддержки языков логического программирования и представления и обработки знаний. Таким образом, осуществляется реализация не только функций ОС в обычном понимании (управление процессами, распределение ресурсов), но и компиляторов, возможно, элементов физического и логического ввода-вывода. В свою очередь это требует разработки принципиально новой элементной базы — полузаказных или полностью заказных СБИС, которые будут элементами параллельных структур на аппаратурном уровне.

Программные средства. Разработка аппаратурного и программного обеспечения ЭВМ пятого поколения включает три фазы. Первая фаза предусматривает создание технологии для разработки аппаратурно-программных средств, вторая — разработку подсистем логического вывода и баз знаний, третья — интегрирование этих подсистем.

Комплекс базовых программных средств включает интеллектуальный интерфейс и подсистемы решения задач, логических выводов и управления базой знаний. Предполагается также создание трех вспомогательных подсистем: интеллектуальной — для разработки программных средств, автоматического проектирования баз знаний и подсистемы проектирования СБИС и архитектуры ЭВМ. Базовые программные средства смогут обращаться к трем универсальным базам знаний (БЗ), организованным как компоненты единой системы управления БЗ. В базу знаний языков входят: основной словарь пользователя, элементарные структуры предложений, словари и правила грамматики для различных предметно-ориентированных языков. В БЗ системы включаются спецификации самой системы, руководство по языку и библиотеки программных модулей. В прикладной базе знаний найдут место основные программные компоненты, компоненты архитектуры и проектирования СБИС. Будет реализована возможность работы с тремя категориями языков программирования: естественный язык (речь и изображения), язык ядра, языки запросов высокого уровня. Только язык ядра системы найдет непосредственное применение при написании программных средств (он будет поддерживать-

ся машинами БЗ), решения задач и получения выводов. Цель состоит в том, чтобы в качестве языка ядра служил язык решателей задач. Программа, написанная на таком языке, включает логические утверждения ограниченной формы. Исполнение этой программы равносильно получению логической дедукции из ее предложений. Другие языки (обращения к базам знаний) должны быть реализованы в терминах языка ядра. Однако от пользователей ЭВМ не потребуется знаний базовых языков (как ПРОЛОГ или ЛИСП), поскольку они будут взаимодействовать на естественном языке.

Языковые средства. Язык ПРОЛОГ в японском проекте принят в качестве языка ядра для ЭВМ пятого поколения. В основе языка лежит принцип резолюций, представляющий собой одну из форм математической логики. Данный принцип позволяет решать задачи с помощью последовательности логических выводов. Основным элементом программы на языке ПРОЛОГ являются предложения Хорна, которые могут быть прямым утверждением (X работает Z часов в неделю) или импликацией (если X работает Z часов в неделю и зарабатывает N руб/ч, то X получает Y).

Из таких утверждений и импликаций можно вывести спецификацию задания по обработке информации или определить запрос на поиск в базе данных. Следовательно, ПРОЛОГ позволяет соединить спецификацию задания и язык программирования, а также язык программирования и язык запросов к БД. Кроме того, корректность спецификаций может быть проверена с помощью формальных методов.

В настоящее время существует ряд версий языка ПРОЛОГ для ЭВМ типа ЕС, DEC, VAX, PC и т. д. Язык ЛИСП может служить примером более ограниченного языка для логического программирования. Основная структурная единица языка — список или строка. Этот язык, позволяющий описывать другие языки, иногда считается метаязыком. Он был применен для описания языка LOGO (обучающих систем) и языков для вычисления значений алгебраических выражений.

Еще одним языком для ЭВМ пятого поколения является ОККАМ, который используется для проектирования ВС в целом с последующей детализацией, особенно для систем с параллельной структурой.

Рассмотренные языки ПРОЛОГ, ЛИСП, DUVAL, VALID, ОККАМ не исчерпывают всего списка разрабатываемых языков, поскольку программирование является фундаментом, на котором новые системные требования приводят к необходимости создания новых языков и новых диалектов старых языков программирования.

Подводя итог, следует выделить составляющие программного обеспечения (рис. 10,3). Их можно определить как базовую программную систему. Будучи ядром, она включает:

системы управления БЗ и интеллектуального интерфейса;

вспомогательные системы интеллектуальной систематизации, которые облегчают работу над проектом и включают три вспомогательных комплекса: интеллектуального программирования, обращения к базе знаний, проектирования СБИС и архитектур ЭВМ;

интеллектуальные утилитные системы, которые облегчают пользование самой ВС и содержат средства, упрощающие перенос программ и БД из других машин, ДС, выдающую пояснительную информацию по всей системе, а также

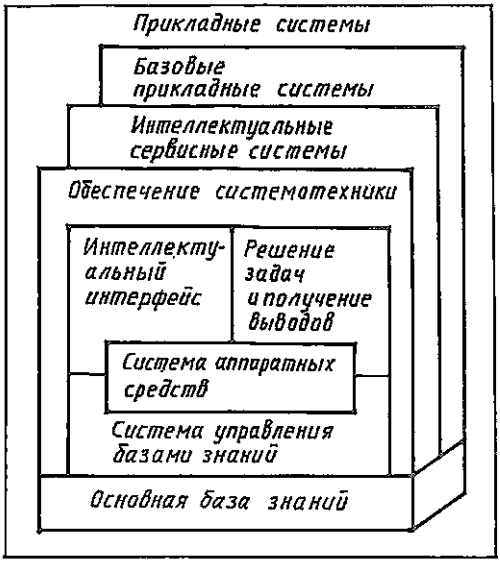


Рис. 10.3.
Система программных средств ЭВМ пятого поколения.

систему контроля и диагностики;

основные базы знаний, к которым могут обращаться прикладные системы, включающие общие знания по языкам взаимодействия, системе и предметным областям;

основные прикладные системы. К ним относятся прикладные системы перевода, понимание речи, интерпретация изображений и образов, консультаций по предметным областям, САПР и другие системы автоматизации.

10.2. ЭЛЕМЕНТЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Направления искусственного интеллекта. Основной особенностью ЭВМ пятого поколения явится широкое использование при организации вычислительных процессов достижений в области искусственного интеллекта (ИИ). Среди направлений ИИ можно выделить следующие: имитацию творческих процессов человека; решение задач, доказательство теорем, обучение; интеллектуализацию решения задач в области САПР, АСУ и создание информационно-поисковых, расчетно-логических и экспертных систем; интеллектуализацию процессов общения с ЭВМ, представления и обработки знаний, планирования решения задач.

Рассмотрим отдельные из этих направлений, связанные с новыми подходами к организации процессов в ЭВМ пятого поколения.

Решение задач. Частным случаем является доказательство теорем. Ход доказательства можно представить в виде замкнутого цикла этапов:

1) исследование состояния решения в данный момент времени; 2) сравне-

ние с тем, что должно быть получено, если отличий нет – завершение доказательств; 3) определение оператора, который мог бы уменьшить существующее различие; 4) последовательное применение операторов, обнаруженных в п. 3, пока не будет найден оператор, который работает. Переход к п. 1.

Чтобы воспользоваться п. 3, необходимо располагать набором эвристических правил, которые связывают имеющиеся операторы с типами различий, выделяемых в п. 2. Необходимо, чтобы в процессе решения был предусмотрен возврат назад в п. 4, когда ясно, что используемые операторы не ведут к цели. Данный подход с некоторыми усложнениями был реализован в первом универсальном решателе задач GPS. В п. 2 имеющийся объект сравнивается с требуемым, при этом может возникнуть не одно различие. В этом случае различия располагаются в порядке, отражающем среднюю трудоемкость снятия различия. Действия в п. 3 реализуются на основании эвристической таблицы, представляющей собой матрицу, строки которой соответствуют типам различий, а столбцы – различным операторам. Метка, стоящая на пересечении i-й строки с j-м столбцом, свидетельствует об использовании i-го оператора для снятия j-го различия.

Следует отметить, что данный метод является воплощением эвристического подхода (анализ цели – средства) в отличие от алгоритмического подхода. Если алгоритм – это набор четко сформулированных операций для выполнения определенной процедуры, то эвристика представляет некоторое произвольное правило или стратегию для ограничения объема поиска в пространствах решений.

Кроме эвристических подходов при решении задачи в пространстве состояний,определяемом совокупностью множеств начальных состояний,операторов отображения одного состояния в другое и конечных состояний, широкое распространение получил *метод резолюций*. Суть этого метода заключается в том, что исходя из некоторого множества предложений (начальных состояний) выводятся другие, показывающие выполнимость или невыполнимость исходных предложений. В исходные предложения включают отрицание того утверждения, которое следует доказать. Возникновение в процессе решения противоречия означает доказательство искомого утверждения.

Метод резолюций является полным в смысле следующей теоремы, доказанной Д. Робинсоном [20]: "Если конечное множество предложений невыполнимо, то опровержение может быть получено за конечное число шагов". Доказательство невыполнимости множества исходных предложений с помощью метода резолюций является правильным. Этот метод был положен в основу внутренних механизмов языка ПРОЛОГ.

Механизмы памяти. Хранение и переработка информации – основа функционирования систем ИИ. В сложных системах ИИ предстоит вести файлы данных для обновления модели окружающего мира, структур самосознания и т. д.

Системы с массовой памятью позволяют хранить единицы информации в конкретных позициях, которые могут быть найдены либо последовательно, либо произвольно. Элементы активной памяти связаны в сеть,и поиск может производиться различными методами, простейший из которых – ассоциативный метод адресации.

Развитие компьютерных моделей памяти, систем с базами знаний или сис-

тем, управляемых набором правил, помогает понять, как механизмы хранения информации могут быть созданы в ИИ.

Переводы. Для того чтобы ЭВМ стали широкодоступными, они должны понимать естественный язык. В настоящее время специалисты по ИИ испытывают затруднения при написании программ, понимающих даже ограниченный язык. Основные работы в этом направлении связаны с машинным переводом. Сейчас насчитывается несколько десятков систем подобного рода.

Была выдвинута гипотеза о том, что процесс регистрации информации распадается на три стадии: в предложении разделяется известная информация и полученная, в памяти определяется адрес известной информации, вновь полученная информация записывается в памяти. На основании этого подхода восприятие окружающего мира и восприятие языка отождествляются с процессами обработки программ вычислительной машиной. ЭВМ сначала транслирует программу во внутреннее представление, а потом ее реализует, используя соответствующие данные. Возможно, процесс обработки предложения человеком включает подобную последовательность из двух фаз: входное предложение переводится в команды мозга, а затем принимается решение о выполнении команды или ответе.

Язык как средство общения весьма важен, благодаря ему становится возможно взаимодействие и появляется критерий для установления факторов понимания. В течение длительного времени проводились работы по построению диалоговых программ. Затем появились работы и программы по общению с базой данных на естественном языке.

М. Минский предложил гипотезу, что все связанное с мышлением, включая язык, возможно, зависит от процессов, управляемых с помощью фреймов, которые содержат ядро и набор секций. Назначение фрейма — представлять некоторую модель процесса, зафиксированную в информации о состоянии последнего.

Распознавание. Следующим направлением в исследованиях по ИИ является распознавание образов. Это связано с построением естественного интерфейса в ЭВМ пятого поколения. Хотя работы в этом направлении дают определенные результаты, их возможности ограничены. Человек зрительно воспринимает информацию и преобразует в понятную для мозга форму с высокой скоростью (4,3 млн бит/с).

Для распознавания символов алфавита часто используют шаблоны, однако это приемлемо, если фигура хорошо определена. Ведутся работы по распознаванию естественных образов, в том числе на основе ассоциативных механизмов памяти. В этих системах связанные сигналы, одновременно генерируемые образом, запоминаются таким способом, что при повторении одного из них порождаются остальные.

Все большее признание получает подход, согласно которому восприятие рассматривается как вычислительный процесс. Выделяется несколько сознательных и бессознательных действий человека, от которых зависит успешность восприятия: выделение некоторых свойств в массиве сенсорных сигналов; выбор свойств, подлежащих группировке в более крупные единицы; принятие решения об игнорировании части свойств; получение логических выводов; интерпретация полученных данных; распознавание указателей на определенную интерпретацию.

10.3. ЭКСПЕРТНЫЕ СИСТЕМЫ

Понятие экспертной системы. Под экспертной системой (ЭС) понимается система, обеспечивающая для конкретной предметной области регистрацию знаний человека и доступ к ним. Существующие ЭС в основном предназначены для накопления знаний и общения экспертов между собой. Неподготовленный пользователь может с ограничением работать с такими системами в силу их специфичности. Однако в перспективе такие системы найдут более широкое применение в связи с появлением ЭВМ пятого поколения (при обеспечении диалога на естественном языке).

На разработку ЭС оказали влияние информационно-поисковые системы, основное назначение которых – извлекать требуемую информацию из большого массива данных. Одной из ключевых проблем является трудность поиска данных по индексным (ключевым) сочетаниям, так как при объединении слов в предложения возможны варианты, трудно выражаемые с помощью булевой или другой формальной логики.

Один из подходов к решению проблемы поиска состоит в разработке модели данных, независимой по отношению к запросам пользователей. На начальном этапе создания ЭС трудности связаны с изложением знаний в логически последовательных выражениях, приемлемых для ЭВМ.

После ввода в ЭВМ знаний эксперта последние могут представляться в древовидной форме в виде аксиомы или правила, которые можно считать узлами дерева. Вершиной служит целевая гипотеза – выражение, характеризующее задачу, обладающее вероятностными свойствами и содержащее значения допустимых ошибок. ЭС может иметь одну главную и несколько второстепенных гипотез.

Характеристики ЭС. Основой ЭС является набор фактов и эвристические приемы (правила), которые хранятся в базе знаний ЭВМ. Управляющая программа применяет эти эвристические приемы для работы с хранимыми знаниями в соответствии с запросами пользователей, причем ход рассуждений может быть раскрыт пользователю, чтобы показать, как система пришла к конкретному выводу.

Каждая ЭС включает три основных элемента: управляющую структуру, базу знаний и ситуационную модель. *Управляющая структура* обращается к информации, содержащейся в базе знаний. *Ситуационная модель* служит для интерпретации текущих данных. В *базе знаний* хранятся все данные, связанные с конкретным приложением. Чем больше информации в базе знаний, тем меньше нагрузка на логику выводов в управляющей системе. Любая ЭС должна развиваться, накапливая опыт, либо непосредственно по мере поступления в нее новой информации, либо косвенно по мере запоминания системой приемлемых выводов. ЭС может работать на нескольких уровнях либо поверхностно при быстром ответе, либо тщательно при сложном анализе.

Применение правил (например, if–then) может способствовать разработке базы знаний, создаваемой с использованием языка ПРОЛОГ. В этом смысле не существует разницы между информацией, основанной на фактах (элемент данных), и правилом (оператором программы). И то и другое может храниться в одной базе знаний и выбираться по мере необходимости. Операторы

ПРОЛОГа имеют отношение как к спецификации программы, так и к самой программе.

Правила. Действующие ЭС построены по принципу накопления знаний (правил). Преимуществом является постепенное накопление знаний, причем в форме, легко выражаемой экспертом. Правила, представляемые в базе, эквивалентны некоторой прикладной программе и могут иметь множество различных форматов; одним из распространенных форматов является следующий : if – условие; then – действие, причем компонента then может представлять выводы, утверждения, вероятности, указания и т. д. Первое из нескольких условий, связанных с этим правилом, определяет, является ли оно потенциально справедливым по отношению к текущему состоянию ситуационной модели. Правило может потребовать выполнения ряда условий, прежде чем действие будет разрешено.

В отдельных ЭС правила могут представлять знания, а метаправила служат для манипулирования правилами. В других ЭС под метаправилами понимаются правила более высоких уровней, соответствующие системным программным средствам традиционных комплексов. В стадии разработки и освоения находится несколько языков, не зависящих от правил.

Существуют три принципа построения систем, основанных на правилах: 1) сверху вниз. При этом система отыскивает среди выводов верхнего уровня истинные. Делается попытка установить соответствие между правой частью правил и поставленной целью. При установлении соответствия компоненты левой части принимаются в качестве новых целей,и поиск продолжается. В результате создается дерево целей; 2) на базе модели. В этих системах основой организации служит модель релевантного мира. Правила, используя вводимые данные, создают и корректируют такую модель, в частности строят временну́ю диаграмму модели. Это позволяет делать прогнозы и выявлять причины связи; 3) в виде классической доски. В этом случае правила образуют источники знаний, отражающих опыт, накопленный в определенных предметных областях. Источники связей взаимодействуют через общую БД. Последняя содержит гипотезы, факты и организована в соответствии с уровнями анализа данных. Гипотезы можно строить с помощью источников знаний.

В настоящее время с точки зрения ориентации экспертных систем выделяются проблемно-ориентированные и проблемно-независимые системы (пустые). Первые ориентированы на конкретную прикладную область, их база знаний наполнена элементами знаний из этой области, правила вывода и язык запросов в основном фиксированы. Примерами таких экспертных систем являются системы области медицины, экономики, управления сложными техническими и социальными системами и т. д.

Проблемно-независимые ЭС включают систему представления знаний, стратегию планирования (набор базовых правил вывода) и стратегию диалога (набор базовых средств организации диалога). Для перехода к проблемно-ориентированной ЭС необходимо заполнить базу знаний, сформировать словари диалоговой системы, настроить планировщик на требуемые правила выводов.

10.4. ПРЕДСТАВЛЕНИЕ ЗНАНИЙ

Основные положения. Обработка данных всегда считалась основной функцией ЭВМ. Но в ЭВМ пятого поколения впервые предложено обрабатывать знания, как и всякую другую информацию. Рассмотрим особенности знаний, которые отличают их от данных.

1. *Интерпретируемость.* Данные в ЭВМ могут содержательно интерпретироваться лишь соответствующей программой. Знания отличаются тем, что в них всегда присутствует возможность содержательной интерпретации.

2. *Наличие классифицирующих отношений.* Несмотря на разнообразие форм хранения данных, ни одна из них не обеспечивает возможности компактного описания всех связей между различными типами данных. При переходе к знаниям между отдельными их единицами могут быть установлены такие отношения, как элемент–множество, тип–подтип, ситуация–подситуация, отражающие характер их взаимосвязи. Это позволяет отдельно хранить информацию, одинаковую для всех элементов множеств.

3. *Наличие ситуативных связей.* Эти связи определяют совместимость отдельных событий или фактов, хранимых в памяти, а также отношения: одновременность, расположение в одной области пространства, нахождение в состоянии взаимодействия и т. д. Ситуативные связи позволяют строить процедуры анализа знаний на совместимость, непротиворечивость и другие, которые трудно реализовать при хранении массивов данных.

Появление знаний как информационных объектов для обработки на ЭВМ определило переход от баз данных к базам знаний. Системы управления последними представляют пользователю более мощные процедуры, чем СУБД. В частности, пользователь может работать не только с имеющимися структурами информации, но и создавать свои, обеспечивать связь между структурами пользователя и структурами, хранимыми в базе знаний. В общем случае база знаний и база данных рассматриваются как различные уровни представления информации, хранящейся в интеллектуальном банке информации.

Основным вопросом при создании баз знаний является выбор способа представления знаний и общения с пользователем. Совокупность модели представления знаний и связанных с ней процедур образуют систему представления знаний.

Разделяют декларативное и процедуральное представление знаний. *Декларативное представление* не содержит в явном виде описания процедур. Это, как правило, множество утверждений, не зависящих от того, где они используются. Моделирование предметной области в такой форме требует полного описания ее состояния. Вывод и поиск решений опирается в основном на процедуры поиска в пространстве состояний.

Процедуральные знания содержат в явном виде описания некоторых процедур, при этом текущее состояние представляется в виде набора специализированных процедур, обрабатывающих определенный участок базы знаний. Это позволяет отказаться от хранения описаний всех возможных состояний, требуемых для построения вывода или решения, и ограничиться хранением начального состояния и процедуры, генерирующей необходимые состояния из начального. Представление знаний в такой форме обеспечивает более быстрый поиск решения по сравнению с декларативным, однако уступает им по воз-

можностям коррекции и накопления знаний.

Наиболее распространенными считаются логические и сетевые модели представления знаний. Основой логических моделей является понятие формальной системы, задаваемой четверкой M = (T, P, A, Ф), где T – множество базовых элементов; P – множество синтаксических правил, позволяющих из T строить правильные выражения; A – множество аксиом; Ф – семантические правила вывода, позволяющие расширять множество аксиом за счет других выражений. Использование логик различного типа при построении синтаксических и семантических правил порождает модели различных типов. Широкое распространение получили *предикатные системы*, особенно после создания мощных процедур вывода на базе метода резолюций, лежащего в основе механизмов языка ПРОЛОГ.

Сетевые модели в отличие от логических предоставляют более широкие возможности для описания сложных структур знаний. Основой этих моделей является сеть, вершины которой отождествляются с некоторыми понятиями, а дуги – соотношениями между этими понятиями. При этом вершины могут иметь собственную внутреннюю структуру. Широкое распространение получили сетевые модели в виде семантических сетей и фреймов.

Фреймы. Одним из возможных путей представления знаний является фреймовая организация. Под *фреймом* понимается модель ситуации реального мира, но не конкретной ситуации, а наиболее характерных, основных особенностей ряда близких ситуаций одного класса. Графически фрейм представляется в виде сети, состоящей из узлов и связей между ними. Каждому узлу соответствует определенное понятие. Это понятие может быть задано в явном виде, а может быть и не задано. Не заданные в явном виде узлы называются *терминалами*.

При возникновении конкретной ситуации из памяти извлекается фрейм, соответствующий данному классу ситуаций, и производится уточнение ситуации путем заполнения терминалов. Каждый терминал может устанавливать условия, которым должны удовлетворять его задания.

Представление лингвистических знаний связано с созданием интеллектуального интерфейса пользователя на базе естественного языка. Реализация этого интерфейса предполагает наличие моделей формального описания языка и алгоритмов обработки текстов, а также программных и (или) аппаратных средств, поддерживающих эти модели и алгоритмы. Перечисленные компоненты определяют систему представления лингвистических знаний, которую можно разделить на две части. Основой модельно-алгоритмической части является теория формального языка. Программно-аппаратная часть выполняет прямой и обратный переводы естественно-языковых текстов на язык внутреннего представления ЭВМ, реализуется лингвистическим процессом. В последнем выделяют описание словарей и алгоритмы анализа и синтеза естественно-языковых текстов.

Языки представления знаний условно можно разделить на три группы: обработки символьной информации ЛИСП, РЕФАЛ; ориентированные на поиск решения в пространстве состояний и доказательства теорем, например PLANER, QLISP; общего назначения, к которым относятся KNL, FNL.

Автоматическое программирование. Частным случаем представления и обработки знаний является автоматическое программирование. Сейчас уже

имеется возможность генерировать программную структуру задачи по ее описанию на проблемно-ориентированных языках (ПРИЗ, СПОРА, ПОЭТ и др.). Для систем автоматического программирования характерно следующее: использование метода спецификаций – описание задач на языке очень высокого уровня, желательно в диалоговом режиме; выходной язык, на котором система по спецификации записывает конечную программу (ЛИСП, ПЛ/1); проблемная область, связанная с предполагаемым применением системы; метод действия. Последний используется при автоматическом программировании при доказательстве теорем, формировании программ, индукции, автоматическом выборе данных. В настоящее время уже функционируют такие системы, как ПРИЗ, СИНТЕЗ, ID3.

Система ПРИЗ разработана в Институте кибернетики АН Эстонской ССР и реализована на больших и микроЭВМ. Она позволяет по описанию задачи на входном языке УТОПИСТ (новый УТОПИСТ–НУТ) синтезировать программу из модулей. Пользователь системы должен правильно описать условия задачи, поскольку синтезатор программ свободен от ошибок. Язык УТОПИСТ включает процедурную часть для описания программ, но его основное назначение – описание понятий и задач. Одним из основных операторов языка является оператор задачи, имеющий следующую форму:

$$\text{НА } Z \text{ ВЫЧИСЛИТЬ } \; Y_1, Y_2, \ldots, Y_m \text{ ПО } X_1, X_2, \ldots, X_n ,$$

где X_1, X_2, ..., X_n – имена понятий, описанных семантической моделью с именем Z, которая называется моделью задачи, причем X_i – входы задачи, Y_1, ..., Y_n – выходы. Модели, на которых данными операторами описываются задачи, хранятся в семантической памяти со структурой, имеющей вид дерева, в вершинах которого расположены понятия, а листья соответствуют элементарным понятиям. Для синтеза модуля решения задачи необходима информация о ее содержании, которая представляется семантической моделью. Эта модель содержит переменные и отношения. Отношения выражают связь между переменными и могут иметь вид программных уравнений, семантической модели, входной структуры данных.

Но впереди еще большая работа по реализации всех концепций организации вычислительных процессов в ЭВМ пятого поколения.

Выводы

1. В исследованиях и разработках, проводимых в области ЭВМ пятого поколения, заключаются перспективы организации вычислительных процессов. На уровне пользователя появятся средства, обеспечивающие интерфейсы на естественном языке и на основе графических образов. Эти интерфейсы реализуются лингвистическими процессорами и рядом подсистем: управления базами знаний, логических выводов, автоматизации программирования. Параллельные аппаратные средства поддерживают отдельные функции для обработки символьной информации, работы с базами данных и знаний, поиска решений, логических выводов.

2. Разработки в области ЭВМ пятого поколения базируются на основных результатах, полученных в области ИИ: решении задач и доказательстве теорем, моделировании механизмов памяти, переводах с одного языка на другой, обучении, распознавании образов.

3. Одним из практических результатов ИИ является широкое использование экспертных систем, включающих, как правило, базу знаний, систему логических выводов, систе-

му планирования и диалоговые средства. Различают предметные экспертные и пустые системы. Если первые ориентированы на конкретные предметные области, то вторые могут настраиваться на проблемные приложения.

4. Важнейшим направлением является представление и обработка знаний, для которых характерны интерпретируемость, наличие отношений и ситуативных связей. Это направление включает проблемы создания средств для манипулирования знаниями, их пополнения, устранения в них противоречий, а также создания языков для представления знаний. С этим направлением связаны проблемы автоматического синтеза программ, частично решенные в отечественных системах ПРИЗ и СПОРА.

Рекомендуемая литература: [11, 16, 20].

Вопросы для контроля

1. Охарактеризуйте основные направления разработки ПО ЭВМ пятого поколения.
2. Что понимается под естественным интерфейсом?
3. В чем отличие баз знаний от баз данных?
4. Как организуются вычисления в системе логических выводов?
5. Как организуется вычислительный процесс в экспертной системе?
6. Охарактеризуйте основные направления исследований в области ИИ.
7. Поясните понятие фреймов.
8. Поясните понятие автоматизации программирования.
9. Определите основные идеи системы ПРИЗ.

ЛИТЕРАТУРА

1. *Авен О.И., Коган Я.А.* Управление вычислительным процессом в ЭВМ. – М.: Энергия, 1978. – 240 с.

2. *Анисимов Б.В., Петров В.Я.* Организация вычислительных процессов в ЦВМ. – М.: Высш. шк., 1977. – 408 с.

3. *Головкин Б.А.* Параллельные вычислительные системы. – М.: Наука, 1980. – 519 с.

4. *Головкин Б.А.* Расчет характеристик и планирование параллельных вычислительных процессов. – М.: Радио и связь, 1983. – 273 с.

5. *Дамке М.* Операционные системы микроЭВМ. – М.: Мир, 1985. – 150 с.

6. *Девис У.* Операционные системы. – М.: Мир, 1980. – 436 с.

7. Диалоговая система коллективного пользования PRIMUS 2.5. Концепции и возможности. – М.: Изд-во МИФИ, 1985. – 26 с.

8. *Змитрович А.И.* Операционные системы. – Мн.: Изд-во БГУ им. В.И.Ленина, 1982. – 257 с.

9. *Квиттнер П.* Задачи, программы, вычисления, результаты. – М.: Мир, 1980. – 422 с.

10. *Кристиан К.* Введение в операционную систему UNIX. – М.: Финансы и статистика, 1985. – 318 с.

11. *Майерс Г.* Архитектура современных ЭВМ. – М.: Мир, 1985. – Т. 1. – 337 с.

12. *Максимей И.В.* Математическое моделирование больших систем. – Мн.: Выш. шк., 1985. – 119 с.

13. *Мартин Дж.* Организация баз данных в вычислительных системах. – М.: Мир, 1980. – 662 с.

14. *Мищенко В.А., Лазаревич Э.Г., Аксенов А.И.* Расчет производительности многопроцессорных вычислительных систем. – Мн.: Выш. шк., 1985. – 208 с.

15. Мультипроцессорные системы и параллельные вычисления / Под ред. Ф.Г. Энслоу. – М.: Мир, 1976. – 387 с.

16. *Назаретов В.М., Ким Д.П.* Техническая имитация интеллекта. – М.: Высш. шк., 1986. – 113 с.

17. Основы теории вычислительных систем / Под ред. С.А. Майорова. – М.: Высш. шк., 1978. – 408 с.

18. *Петерсон Дж.* Теория сетей Петри и моделирование систем. – М.: Мир, 1984. – 264 с.

19. *Прангишвили И.В.* Микропроцессоры и локальные сети микроЭВМ. – М.: Энергия, 1984. – 272 с.

20. *Симонс Дж.* ЭВМ пятого поколения: компьютеры 90-х годов. – М.: Финансы и статистика, 1985. – 173 с.

21. Система виртуальных машин для ЕС ЭВМ / Под ред. Э.В. Ковалевича. – М.: Финансы и статистика, 1985. – 360 с.

22. Система малых ЭВМ и их применение / Под ред. Б.Н. Наумова. – М.: Статистика, 1980. – 231 с.

23. *Хокни Р., Джессхоуп К.* Параллельные ЭВМ. – М.: Радио и связь, 1986. – 390 с.

24. *Хромов В.И., Ульянов С.А.* Введение в программирование для систем телеобработки данных. – М.: Финансы и статистика, 1982. – 265 с.

25. *Хусаинов В.С.* Программирование ввода-вывода в ОС ЕС ЭВМ на языке Ассемблера. – М.: Статистика, 1980. – 264 с.

26. *Цикритзис Д., Бернстайн Ф.* Операционные системы. – М.: Мир, 1977. – 336 с.
27. *Шаньгин В.Ф., Костин А.Е.* Организация вычислительных процессов на микроЭВМ. – М.: Высш. шк., 1984. – 119 с.
28. *Шоу А.* Логическое проектирование операционных систем. – М.: Мир, 1980. – 360 с.
29. *Шрайбер Т.Д.* Моделирование на GPSS. – М.: Машиностроение, 1980. – 506 с.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ОГЛАВЛЕНИЕ

Учебное издание

Вишняков Владимир Анатольевич

ОРГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ ЭВМ И СИСТЕМ

Заведующий редакцией *А.Ф. Зиновьев*
Редактор *С.С. Голод*
Младшие редакторы *С.А. Когадеева, Т.И. Крючкова*
Художник обложки *С.В. Баленок*
Художественный редактор *Ю.С. Сергачев*
Технический редактор *Л.И. Счисленок*
Корректоры *Р.К. Логинова, Г.В. Вагабова*
Оператор *А.И. Маль*

ИБ № 2641

Подписано в печать с оригинала-макета 03.10.88. АТ 12691. Формат 60 x 90/16. Бумага офсет. Гарнитура Пресс Роман. Печать офсетная. Усл. печ. л. 13. Усл. кр.-отт. 13. Уч.-изд. л. 16,08 . Тираж 9500 экз. Заказ 6068. Цена 85 к.

Издательство "Вышэйшая школа" Государственного комитета БССР по делам издательств, полиграфии и книжной торговли. 220048, Минск, проспект Машерова, 11.

Типография "Победа". 222310, Молодечно, ул. Тавлая, 11.