

**БИБЛИОТЕЧКА  
ПРОГРАММИСТА**

Л. ДЖ. КОЭН

# Анализ и разработка операционных систем



# OPERATING SYSTEM ANALYSIS AND DESIGN

by  
Leo J. Cohen

SPARTAN BOOKS

---

NEW YORK • WASHINGTON

БИБЛИОТЕЧКА  
ПРОГРАММИСТА

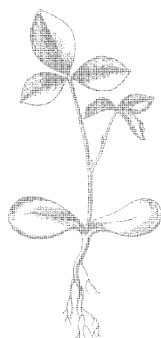
---

Л. Дж. КОЭН

# АНАЛИЗ И РАЗРАБОТКА ОПЕРАЦИОННЫХ СИСТЕМ

Перевод с английского  
под редакцией  
В. Ф. ТЮРИНА

ИЗДАТЕЛЬСТВО «НАУКА»  
ГЛАВНАЯ РЕДАКЦИЯ  
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ  
МОСКВА 1975



Scan AAW

**Анализ и разработка операционных систем.** Л. Дж. Коэн. Перевод с английского В. П. Варакина и М. К. Иванова под редакцией В. Ф. Тюрина. Изд-во «Наука», Главная редакция физико-математической литературы, 1975.

Автор книги — специалист в области разработки математического обеспечения. Одна из ее задач состоит в том, чтобы дать ясное представление о том, какова роль операционной системы в общей системе математического обеспечения. В этой книге выражен подход к разработке операционных систем, отличный, например, от подхода фирмы IBM.

Книга может быть полезна широкому кругу специалистов, интересующихся операционными системами.

## ОГЛАВЛЕНИЕ

Предисловие редактора перевода . . . . .	7
Г л а в а 1. Введение . . . . .	9
Ассемблеры (10). Управление пакетом (11). Компиляторы (13).	
Г л а в а 2. Цели проектирования ОС . . . . .	16
Введение (16). Критерии работы (17). Внутренние цели проектирования (20). Исполнитель (первое определение) (22). Внешние цели проектирования (23). Операционная система (первое определение) (25). Логическое разделение программ с помощью аппаратуры (27). Прерывания (28). Защита и распределение памяти (30).	
Г л а в а 3. Некоторые общие соображения . . . . .	32
Квантование времени (34). Мультипрограммирование запросов на ввод-вывод (37). Небуферируемый запрос на ввод-вывод (39). Буферируемый запрос на ввод-вывод (40). Замечания (42).	
Г л а в а 4. Исполнитель небуферируемого запроса на ввод-вывод . . . . .	44
Общее описание (44). Состояния программы (45). Приоритет (50). Запрос на ввод-вывод (53). Завершение ввода-вывода (57). Завершение выполнения программы (60). Общие замечания (61).	
Г л а в а 5. Запрос центрального процессора . . . . .	63
Введение (63). Поля запросов (64). Слово запроса (66). Запрос ЦП (67). Поле ЦП (67). Функция ОС (70). Модуль возврата (72). Схема запросов (74). Схема запросов для монитора (MPX) (74). Общие замечания (81).	
Г л а в а 6. Основные модули ОС . . . . .	83
Средства вычислительной системы (84). Общие замечания (85). Приемник (88). Директор (90). Загрузчик (92). Монитор (MPX) (94). Распреде-	

литель (94). Диспетчер (94). Взаимосвязи между модулями ОС (97). Запросы (100). Модуль ввода (101). Исполнительный модуль (103). Модуль вывода выходной информации (104). Общие замечания (106). Динамические взаимосвязи (107).	
<b>Глава 7. Операционная система для небуферизуемых работ . . . . .</b>	<b>112</b>
Введение (112). Постановка задачи (113). Конфигурация аппаратного обеспечения (114). Модуль ввода (114). Модуль вывода (117). Монитор (MPX) (118). Блок-схема монитора (119). Приемник (120). Директор (126). Загрузчик (133). Блок окончания задач (137). Распределитель (142). Диспетчер (145).	
<b>Глава 8. Работа с буферизацией и запрос ввода-вывода . . . . .</b>	<b>149</b>
Введение (149). Последовательные файлы (150). Буферы ввода (151). Буферы вывода (152). Первый тип ввода-вывода (152). Ввод-вывод 2-го типа (154). Общие замечания (155). Запрос ввода-вывода (156). Возникновение запросов ввода-вывода (157). Поле запроса ввода-вывода (158). Расширение поля запроса ЦП (161). Состояние буфера ввода первого типа (163). Буфер ввода первого типа (163). Буфер вывода первого типа (166). Буфер второго типа (166). Заключение (170).	
<b>Глава 9. Модель монитора (MPX) . . . . .</b>	<b>171</b>
Введение (171). Описание состояния ЦП (171). Операторы перехода (172). Буфер (один/два) (173). Копировать (очередь/мя) в (очередь/мя) (174). Цикл (174). Уничтожить (очередь/мя) (174). Увеличить счетчик буфера (174). Сравнить (AT/IOT) с (очередь/мя) (175). Режим ОС (175). Поместить (AT/IOT) в (очередь/мя) (175). Поместить IOT в (очередь) к каналу (176). Возврат (COT/AT) (176). Выбрать (AT/IOT) из (очередь/мя) (176). Выбрать IOT из (очередь) к каналу (176). Начать ввод-вывод (177). Перевести указатель (очередь) (177). Проверить тип буфера (177). Проверить IOT (канал/устройство) (177). Модель монитора (177). Запрос ввода-вывода (178). Прерывание по окончании ввода-вывода (183).	
<b>Сокращения . . . . .</b>	<b>190</b>

## ПРЕДИСЛОВИЕ РЕДАКТОРА ПЕРЕВОДА

Развитие электронных вычислительных машин (ЭВМ) и их программного обеспечения за последние 20 лет проходило очень интенсивно. Частью программного обеспечения ЭВМ являются операционные системы (ОС), которые также прошли путь бурного развития.

Основные функции ОС:

- ввод задачи пользователя в ЭВМ;
- управление ходом ее решения;
- обработка аварийных ситуаций;
- распределение ресурсов ЭВМ между задачами пользователя;
- защита программ и данных, принадлежащих различным задачам пользователя, от взаимного влияния, и т. д.

ОС — это основа программного обеспечения современной ЭВМ, определяющая обстановку, в которой работают обслуживающий персонал, пользователи, руководство вычислительного центра. ОС оказывают огромное влияние на развитие как аппаратных возможностей ЭВМ, так и систем программирования.

Существующие в настоящее время операционные системы отличаются очень большим разнообразием возможностей, степенью универсальности, структурой организации, алгоритмами функционирования.

Предлагаемая вниманию читателя книга является попыткой автора систематизировать множество методов реализации, некоторые оценочные данные, а также функциональное строение ОС. Она дает достаточно полное представление о приоритетном планировании, буферизации ввода-вывода, мультипрограммировании и т. д. Основной акцент книги направлен на разработку



ОС. Делается анализ построения ОС, описываются основные блоки, приводится подробный анализ частей ОС. Эта книга — одна из немногих, которые помогают разобраться в принципах построения ОС. В основном в ней делается обзор систем пакетной обработки данных. Данная книга не свободна и от недостатков: очень много повторений; иногда приводятся слишком развернутые пояснения и примеры, а в некоторых местах — наоборот, примеров явно не хватает.

Роль ОС в наши дни бурно возрастает, и число читателей, интересующихся ими, постоянно увеличивается. Поэтому выход этой книги можно считать весьма своевременным.

Перевод выполнен В. П. Варакиным и М. К. Ивановым.

*В. Ф. Тюрин*

## ГЛАВА 1

### ВВЕДЕНИЕ

В этой книге описываются операционные системы (ОС), и поэтому естественно начать с определения этого термина. Обычно используемое формальное определение многословно и трудно применимо для большинства реальных ситуаций.

Возможно, что основным источником этих трудностей в определении термина *операционная система* является сама вычислительная машина. Назначение ЭВМ, ее центрального процессора (ЦП), ее памяти и периферийных устройств состоит в выполнении программ. Некоторые из них написаны пользователями вычислительной системы и называются *прикладными программами*. Другие программы должны быть уже написаны программистами к моменту создания прикладных программ. Они в некотором смысле являются частью самой вычислительной системы точно так же, как магнитные ленты и ЦП являются частями этой вычислительной системы (ВС). Для описания этих программ служит термин *программное \*) обеспечение* системы. Операционная система также является программой, и в настоящее время, пожалуй, самой большой программой, выполняемой ЭВМ. Она состоит из совокупности программ, являющихся частью программного обеспечения.

---

\*) Программное обеспечение системы называют также математическим обеспечением ЭВМ. (Прим. ред.)

Эти термины годятся для разговора на высоком уровне, но мало что дают для понимания роли того, что мы называем операционной системой, сопоставляя ее с программным обеспечением и прикладными программами. Именно поэтому целью настоящей работы является разработка формализации, которая послужит средством для четкого выделения операционной системы среди всех программ, выполняемых данной ЭВМ. Пусть читатель нас извинит за то, что мы все же будем пользоваться термином «операционная система», пока не давая его определения. Главы 2, 3 и 5 внесут ясность в эту ситуацию, а в главе 6 будет описан основной набор функциональных блоков, входящих в любую операционную систему. Во введении будут рассмотрены только общие характеристики и то, с чего начали свое развитие операционные системы.

## **Ассемблеры**

Возможно, что наиболее ранней системой программирования для пользователей была система обобщенного программирования UNIVAC (GP) для ЭВМ UNIVAC I, а позднее для UNIVAC II. Основной характеристикой языка ассемблера для этих машин было представление данных и ячеек памяти в виде символических ссылок. С точки зрения пользователей это было удобнее написания программ в абсолютных адресах. Кроме того, этот ассемблер давал пользователю доступ через инструкции ко всему имеющемуся оборудованию, не ограничивая доступ пользователя к оборудованию по сравнению с программированием в кодах.

Короче говоря, на этой стадии развития ассемблер был «программным обеспечением» для вычислительной системы. Конечно, трудно представить себе такое программное обеспечение в качестве операционной системы, но расширение возможностей ассемблера можно несомненно считать положительным шагом в направлении развития ОС.

Ранние версии языка UNIVAC (GP) представляли пользователю ограниченные средства для модификации операндов и вычисления адресов как во время компиляции, так и при выполнении рабочей программы. Это

поразительно увеличило гибкость программирования на GP и стало фактически стандартной характеристикой всех ассемблеров. Дальнейшим развитием стало введение псевдокоманд, не входящих в систему команд ЭВМ, ставших мощным средством для создания сложных программ. Эти псевдокоманды переводились транслятором в соответствующую последовательность машинных команд. Таким образом, развитие ассемблеров происходило по пути предоставления пользователям больших возможностей, не сокращая в то же время доступ к возможностям вычислительной системы.

### Управление пакетом

На этой стадии развития ассемблер предоставляет пользователям большие возможности для решения сложных задач программирования. Результатом работы ассемблера является последовательность машинных команд, получавшаяся на каком-либо носителе (например, на магнитной ленте или перфокарте). Эти машинные команды являются результатом трансляции мнемонических, псевдо- и макрокоманд. Сюда входят и подпрограммы, извлекаемые из библиотеки на магнитной ленте во время процесса сборки и вставляемые в программу пользователя в соответствии с наличием в ней псевдокоманд, специально предназначенных для этих целей. Кроме того, ассемблером генерируется последовательность команд, дающих возможность загрузить собранную программу в систему для выполнения счета по ней.

Все возрастающее число пользователей хотело использовать вычислительную систему для выполнения сборки и счета программ. Поэтому вполне естественным стало и развитие понятия *работы и разработки методов управления* последовательностью работ в пакете, состоящем из произвольно чередующихся работ по сборке и выполнению программ. Для выполнения такого управления требовалась специальная программа, представляющая собой единое целое с оборудованием вычислительной системы. Это было, конечно, большим шагом вперед по сравнению с ассемблером, и этим шагом ознаменовался переход к периоду *программного обеспечения* вычислительной системы (BC).

Есть определенная характеристика этого раннего типа программного обеспечения, которую следует здесь отметить. Во время выполнения любой работы в пакете, будь то сборка или выполнение готовой программы, никакая даже самая небольшая часть программного обеспечения не находилась в оперативной памяти системы. Это значит, что практически вся память предоставлялась в распоряжение текущей работы. На самом деле, необходимость в отведении минимального набора системных возможностей для программного обеспечения была. Это продиктовано тем, что во время выполнения работы программное обеспечение должно было перекачиваться из оперативной памяти на какой-либо движущийся носитель. Это выполнялось следующим способом: при завершении выполнения текущей работы загружалось программное обеспечение и ему передавалось управление. Далее эти системные программы принимали вводную информацию следующей работы, которая должна быть выполнена, определяя при этом, является ли это сборка или выполнение готовой программы. В случае сборки начиналась загрузка программы ассемблера, обычно находящаяся на том же внешнем носителе, что и программное обеспечение. После загрузки ассемблер начинал обрабатывать предложенную ему программу. По окончании работы ассемблера опять начиналась загрузка системных программ, и этот процесс продолжался до тех пор, пока не обрабатывался весь пакет. Если системной программой выяснилось, что следующая работа в пакете — счет по готовой программе, то начиналась загрузка этой программы и затем ей передавалось управление. После окончания работы этой программы опять загружались системные программы, и процесс продолжался, как это уже описывалось.

Такое последовательное выполнение пакета работ, чередующееся с работой системных программ, называлось *системой управления пакетом*.

Хотя программы управления пакетом и не присутствовали в памяти машины во время выполнения работ пакета, ясно, что они накладывали отпечаток и на сами рабочие программы. Во-первых, ассемблер должен уметь загружать программы управления пакетом после окончания операций сборки. Во-вторых, эта функция

должна проявляться и в каждой рабочей программе в виде набора команд, вставленного в соответствующее место в программе. Таким образом, мы видим, что на работу ассемблера оказывает влияние система управления пакетом.

Однако третье, и для нас наиболее важное влияние, состоит в следующем. Система управления пакетом требует, чтобы информация, позволяющая определить тип работы, предшествовала каждой работе в пакете. Эта информация указывает на сущность работы и для системы управления пакетом служит указателем того, какие действия должны быть предприняты для начала выполнения работы. Форма описания этой информации обычно рассматривается как *язык запроса работы*, и на примере всех современных вычислительных систем можно видеть, что этот язык существенно отличается от языка, на котором пишутся программы. Важным различием этих двух языков является то, что назначение языка программирования состоит в управлении ассемблером во время сборки и вычислительной системой во время работы программы, а язык запроса работы предназначен для управления работой системных программ.

## Компиляторы

Язык компилятора отличается от языка ассемблера, так же, как проблемно-ориентированный язык отличается от машинно-ориентированного языка. Как было отмечено ранее, машинно-ориентированный язык предназначен для того, чтобы дать пользователю доступ ко всем средствам ВС. Таким образом, программной единицей самого низшего уровня здесь является мнемоническая команда. С другой стороны, проблемно-ориентированный язык предназначен для того, чтобы дать пользователю набор логических функций, которые могут быть более или менее тесно связаны с блок-схемой решения задачи. С неформальной точки зрения операторы языка компилятора могут рассматриваться в качестве основных программных элементов.

Возможно, наиболее очевидное влияние языка компилятора на ОС сказывается на выполнении операций ввода-вывода. Обычно пользователю предоставляется в

распоряжение набор операторов для чтения и записи файлов данных, находящихся во внешней памяти. В языке компилятора обращение пользователя к данным, подлежащим передаче, является обычно символическим и выражается в терминах структур и форматов. Программирование же в командах, необходимое для записи или извлечения данных из оперативной памяти машины, для выполнения операций ввода-вывода или передачи данных на периферийные устройства, организовано так, что его лучше выполнить в виде специальной подпрограммы. Более того, такие подпрограммы могут быть сделаны стандартными для всех пользователей данной ЭВМ, что предпочтительнее генерации подпрограмм для ввода-вывода при каждой компиляции. При загрузке программы могут быть вызваны в оперативную память необходимые ей подпрограммы в виде стандартного пакета.

Тогда становится понятной и основная обязанность компилятора на этом этапе, которая состоит в том, чтобы вставить в программу вместо оператора ввода-вывода звено в машинных командах. Это звено устанавливает связь программы пользователя со стандартными программами обработки ввода-вывода при выполнении программы. В современных ОС эти стандартные программы собраны в пакет ввода-вывода и, естественно, приводят к необходимости обработки прерываний.

Появление пакета ввода-вывода относится к 1958 году. Оно вызвало лавинообразное развитие программного обеспечения, которое вскоре привело к тому, что мы сейчас называем третьим поколением ЭВМ. Этот процесс продолжался не затихая, и то, что вначале было желательным, затем превратилось в поток, угрожающий смыть программиста, в нарастающий прилив всевозможных инструкций и технического жаргона.

Сейчас уже можно указать в этой области полезные идеи и методы.

Цель данной книги и состоит в том, чтобы определить основные свойства операционных систем и найти средства для их формального анализа и разработки. Основным понятием является *мультипрограммная операционная система*. Оставляя формальное определение этого понятия до следующих глав, скажем, что мультипрограммирование означает выполнение сразу

нескольких, а не одной программы на ЭВМ. Чтобы выполнить это, необходимо программное обеспечение, которым и является операционная система. Во всех главах книги мы будем уделять наибольшее внимание тем программам, которые служат этой цели. Такие системные программы, как программы сортировки, ассемблер, компилятор, в этой книге не рассматриваются, хотя это не означает, что такие программы не играют никакой роли. На самом деле они очень важны в общем составе программного обеспечения. Но мы не хотим загромождать рассмотрение основного предмета данной книги, а именно — мультипрограммных операционных систем.



## ГЛАВА 2

### ЦЕЛИ ПРОЕКТИРОВАНИЯ ОС

#### Введение

Вычислительная система представляет собой совокупность сложных логических, механических и электрических элементов. Ее характеристики являются взаимной функцией всех элементов совместно, и следовательно, значение термина *характеристика системы* всецело зависит от круга интересов человека, использующего его. Например, для инженера, работающего с периферийным оборудованием, коэффициент готовности является существенным ингредиентом характеристики системы. Для пользователя вычислительной системы, в частности, для программиста, который использует язык высокого уровня, характеристика системы заключается в легкости доступа к файлам и устройствам, точности и полноте сообщений об ошибках и времени выполнения работы.

С другой стороны, для персонала управления системой характеристикой системы является степень использования различного оборудования, причем этот персонал должен оценивать систему и с точки зрения пользователя.

Совершенно ясно, что для заданной конфигурации ЭВМ смысл, который вкладывает инженер по аппаратному обеспечению в понятие характеристики системы, является поистине академическим в сравнении с понятиями характеристик системы, которые имеют значение для пользователей и персонала управления системой.

Пользователи системы должны принять и во многих случаях примириться с характеристиками аппаратного обеспечения, если они не становятся настолько бездонными, что проверка и тестирование грозит нанести серьезный урон характеристикам системы с любой точки зрения.

Однако нашей задачей является рассмотрение операционной системы и связей, которые она формирует между программами пользователей и логическими возможностями аппаратного обеспечения данной системы. Предполагая, что характеристики аппаратного обеспечения удовлетворительны, обратимся к вопросам, касающимся характеристик системы с точки зрения персонала, управляющего системой, и пользователя.

### **Критерии работы**

Необходимо, чтобы в течение заданного периода работы системы, скажем, в течение часа, центральный процессор был бы загружен точно 3600 секунд. В течение этого периода работы 3600 секунд центрального процессора будут распределены между прикладными программами и программами операционной системы, которые способствуют выполнению первых. Все то время центрального процессора, которое не может быть распределено вышеуказанным образом, в дальнейшем будем называть *свободным временем*. Это общее время простоя центрального процессора в течение заданного периода, когда он ничего не выполняет и ожидает прихода следующей работы.

Общее число секунд центрального процессора, отдаваемых на выполнение прикладных программ в течение заданного периода функционирования системы, в дальнейшем будем называть *временем загрузки центрального процессора прикладными программами*.

Следует заметить, что если заданный набор прикладных программ должен пройти через вычислительную систему с момента начала и до конца работы, то каждая программа будет требовать, чтобы ей отдали процессор на определенное время. Если заданный набор прикладных программ должен пройти через систему в течение некоторого периода времени, то этот набор определяет

время загрузки центрального процессора прикладными программами за этот период.

В предыдущих разделах мы подразумевали, что все программы, использующие вычислительную систему, делятся на две категории.

К первой категории мы относили прикладные программы, все остальные — программы операционной системы. Такое определение для операционной системы вряд ли можно назвать удовлетворительным, однако при рассмотрении критериев работы системы этого более чем достаточно. Благодаря тому, что время центрального процессора фиксируется, в течение рассматриваемого периода работы системы все три категории времени: время работы прикладных программ, время работы программ операционной системы и свободное время легко отличить друг от друга. Следовательно, если не выполняется ни одна прикладная программа и система не простаивает, т. е. работа все же выполняется, то те программы, которые в это время работают, относятся к операционной системе. Выполнение программ операционной системы требует использования ЦП и, следовательно, если можно так выразиться, также увеличивает загрузку ЦП.

В предположении, что за вычислительную систему должны платить ее пользователи, резонно предположить, что за ту часть времени, которая будет затрачиваться на выполнение прикладных программ, они заплатят пропорционально времени, входящему в общее время загрузки ЦП прикладными программами. Кроме того, они совместно должны расплачиваться за использование операционной системы, т. е. выполнение прикладных программ связано с общей «граничной» стоимостью, которая устанавливается в форме стоимости использования ЦП операционной системой. Этим обуславливается то, что загрузку ЦП операционной системой называют «граничной», «верхней» или «верхней системной». Конечно, существование свободного времени подразумевает некую загрузку ЦП, за которую пользователи должны платить меньше или не платить вовсе, что, в свою очередь, несколько уменьшает общую стоимость загрузки ЦП операционной системой.

Индивидуальная загрузка ЦП прикладными программами данного пользователя, обусловленная вы-

полнением их в системе, не меняется от числа программ других пользователей, использующих систему. Время же, в течение которого пользователь ожидает завершения работы, может, конечно, меняться, как функция от числа прикладных программ, обращающихся к средствам системы. Нужно отличать это время от времени ЦП, которое может потребоваться для выполнения прикладной программы. При реализации оживленного мультидоступа, где средства системы должны запрашиваться попеременно всеми прикладными программами, задержки в выполнении отдельной программы будут появляться с большей вероятностью. В результате этого время обслуживания работы может меняться весьма нежелательно, как функция от числа конкурирующих пользователей системы.

Таким образом, вычислительная система может обеспечить своим пользователям удовлетворительное время обслуживания при обременительной верхней стоимости.

Предположим такую ситуацию, при которой свободное время имеется в наличии, и новые пользователи ставятся системой на обслуживание. Возможным следствием этого было бы возрастание верхней граничной стоимости за счет увеличения активности системы до такой степени, что эта стоимость превысит сумму вкладов пользователей. Дополнительные задержки, вызванные увеличением обменов за счет числа прикладных программ, могут настолько удлинить индивидуальное время обслуживания, что граничная стоимость становится недопустимой.

Другим полезным критерием работы системы является нагрузка памяти. Предположим, что единицей памяти является слово и что в памяти можно разместить прикладные программы размером, скажем, в 40 К слов. Тогда за час память обеспечивала бы для прикладных программ  $40 \text{ К} \times 3600$  слов-секунд. Если отдельная прикладная программа использовала бы 3000 слов оперативной памяти, и время ее обслуживания было бы равно двум минутам, то нагрузка памяти была бы равна  $3000 \times 120$  слов-секунд. Нагрузка памяти — хороший критерий, который можно использовать для связи «внутренней характеристики» (эффективного времени) с «внешней характеристикой» (временем

обслуживания, turn around time). Каждая прикладная программа во время ее выполнения находится в памяти, занимая необходимое количество слов и создавая загрузку памяти. Таким образом, каждая отдельная загрузка памяти выражается произведением объема занимаемой памяти на время обслуживания.

Поскольку при обработке каждой отдельной программы на объем доступной к использованию памяти накладывается ограничение, то повышение эффективности работы системы возможно за счет неизменного сохранения нагрузки памяти.

Такое сохранение загрузки памяти улучшает в общем случае время обслуживания.

### **Внутренние цели проектирования**

На владельцев или персонал управления вычислительной системой возложена финансовая ответственность за эту систему. Именно, руководство системой устанавливает размер ежемесячной платы.

Если всего несколько пользователей создают в отдельности малые нагрузки на ЦП, тогда будет существовать совокупное количество свободного времени. Но пользователи системы вряд ли примирятся с тем, чтобы платить за все время работы системы. Задачей руководства является определение объема свободного времени и его продажа новому пользователю.

Возрастание загрузки ЦП прикладными программами соответствует уменьшению свободного времени и увеличению верхнего граничного времени. Это предполагает, что основной внутренней целью или целью руководства системы является распределение средств ЭВМ, ее ЦП, памяти и периферийных устройств по различным прикладным программам так, чтобы уменьшить при этом верхнюю граничную стоимость.

Система должна иметь возможность управлять параллельной обработкой как можно большего числа прикладных программ при возможно меньшей стоимости функционирования или верхней граничной стоимости.

Иначе можно сказать следующим образом: главной внутренней целью операционной системы является максимальное использование ЦП прикладными программами. Этот вопрос отчасти решается увеличением числа

программ, параллельно обрабатываемых в системе. При этом снижается вероятность такой ситуации, когда обрабатываемые программы используют средства системы, отличные от ЦП, такие, как каналы, внешние устройства, и т. д., и ЦП относительно долго простаивает.

Таким же важным вопросом является и внутренняя структура системы, в частности, та ее часть, которую можно было бы назвать исполнительной. Здесь существенное значение имеет то, насколько сжата и легко доступна информация, с которой система работает при организации выполнения нескольких прикладных программ. Это существенно влияет на загрузку ЦП операционной системой. При минимизации загрузки ЦП операционной системой освобождается больше свободного времени и соответственно растет время ЦП, которое может быть отдано пользователям. Общее время ЦП, которое входит в загрузку ЦП прикладными программами за рассматриваемый период, мы в дальнейшем будем называть *пропускной способностью* в этот период.

Существует другая внутренняя цель для создания операционной системы, которая непосредственно кроется в заинтересованности руководства в пропускной способности, а именно: увеличению пропускной способности всей системы периферийных устройств.

О выполнении прикладной программы, вероятно, думают как об операции передачи файлов, трансформирующей определенные входные файлы в определенные выходные файлы и включающей в себя время задержки для обработки этих данных в течение их передачи. Если это представлять себе так, то выполнение прикладной программы не закончится до тех пор, пока не завершатся все операции передачи файлов, которые инициирует данная программа, так что задержка в передаче файлов, обусловленная, скажем, переполнением канала при выполнении нескольких параллельно обрабатываемых прикладных программ, будет задержкой в обслуживании программы. Следовательно, эта программа будет находиться в памяти больше времени, занимая ячейки памяти в течение указанного дополнительного промежутка времени, тем самым увеличивая нагрузку на память и создавая напряженную ситуацию в системе.

Ясно, что это находится в противоречии с желанием руководства увеличить число пользователей системы,

так как, чем в этом случае будет больше число пользователей, тем больше вероятность того, что средства ввода-вывода станут менее доступны и, следовательно, индивидуальное время обслуживания увеличится. Здесь мало чем может помочь система, которая «пытается» разрешить это противоречие, уменьшая время работы всех периферийных устройств при их максимальных скоростях и распределяя устройства и доступы к ним таким образом, чтобы уменьшить суммарное время пуска/останова (магнитных лент, АЦПУ и устройств ввода) или сократить чрезмерное количество перемещений головок (для устройств памяти с произвольным доступом). Наряду с этим операционная система должна заботиться о том, чтобы периферийные устройства, закрепленные за данными прикладными программами, оставались свободными, так как эти прикладные программы не обработаны еще ЦП, чтобы использовать устройства ввода-вывода.

Из вышесказанного следует, что одной из наиболее важных внутренних целей руководства, которые оно преследует при разработке операционной системы, является увеличение пропускной способности устройств и распараллеливание работы внешних устройств. Эта задача является центральной темой следующей главы. Главы 4 и 9 посвящаются разработке методов, с помощью которых будут достигнуты те внутренние цели, которые рассматривались в этом разделе.

### **Исполнитель (первое определение)**

Если вычислительная система должна совершенствоваться в соответствии с целями, к которым стремится руководство, этот вопрос должен разрешаться с помощью создания программы. Несмотря на то, что весьма большое количество описанных выше критериев подходит для так называемых внутренних целей разработки системы, ясно, что программа, призванная удовлетворять этим целям, имеет хорошо поставленную для выполнения задачу, т. е. от такой системной программы требуется, чтобы она могла управлять отбором тех прикладных программ, которые должны обрабатываться ЦП. Кроме того, она должна управлять порядком распределения периферийных устройств между при-

кладными программами, старающимися одновременно получить доступ к ним. Эту программу в операционной системе мы будем называть *исполнителем*. При этом надо понимать, что такая исполнительная программа является лишь одной из программ, составляющих операционную систему. Здесь и в дальнейшем исполнителем будем называть те программы операционной системы, которые предназначены выполнять две функции, описанные выше, а именно, распределять ЦП и устройства ввода-вывода между несколькими прикладными программами, резидентными в системе.

Несколько позднее мы определим логическую сущность того, что называется *запросом* (transaction).

Запрос представляет в операционной системе прикладную программу. Фактически этот идентификатор, используемый операционной системой вообще и исполнительной программой в частности для того, чтобы зафиксировать пользователя и его программу на время ее обработки.

Тогда можно было бы сказать, что исполнителем является та программа, которая манипулирует запросами, т. е. выполнением прикладных программ, так, чтобы удовлетворить требованиям минимальной верхней гарантийной стоимости, максимального использования ЦП, максимальной пропускной способности устройств и максимальной загрузки периферийных устройств.

## **Внешние цели проектирования**

Как уже отмечалось ранее, те критерии, которые нужны пользователю при оценке работы системы, являются качественными и в значительной мере более субъективными, нежели критерии руководства. В частности, его интересует набор языков, необходимый для ввода его работ в систему, и вид, в котором он получает выходные данные, а также то, с какой скоростью это происходит. Пользователь системы интересуется тем, какое среднее время необходимо для выполнения его работ.

Ясно, что оценка работы системы с точки зрения пользователя, даже если ее трудно выяснить, имеет существенное значение для руководства системы, ибо неудовлетворенный покупатель больше не купит того, что ему не понравилось. Работа системы, если смотреть



на нее глазами пользователя, наводит на мысль о существовании некоторых внешних целей, которых можно достигнуть при проектировании операционной системы.

Для того, чтобы пользователи могли иметь возможность инициировать свои задачи в вычислительной системе и получать носители с выходными данными или листинги, требуются, очевидно, некоторые другие программы. Эти программы должны распознавать работу пользователя на устройстве ввода, например, на устройстве считывания с перфокарт.

Они затем введут описание работы на языке управления работами, интерпретируя это описание в набор программ, которые должны будут загружаться в таком порядке, чтобы выполнить требуемую работу. Эта последовательность событий вообще будет описываться как функция приема операционной системы. Если подойти к этому с другой стороны, то операционная система, как набор программ, будет включать в себя и программы, предназначенные выполнять логические операции, которые требуются для выполнения этой функции приема.

Очевидно, чем быстрее реакция на сообщение о присутствии задачи, тем лучше. Обычно пользователь предпочитает увидеть запрос на свою работу считанным как можно раньше, даже если работа ожидает приема во внутренней очереди. Альтернатива заключается в том, чтобы запрос на работу ждал за пределами системы, на виду, до тех пор, пока система не будет готова принять его.

Следующим шагом после принятия запроса будет трансляция с языка запроса работы на машинный язык. Трансляция должна производиться независимо от того, будет ли язык запроса на работу простым или сложным.

Время выполнения программ трансляции заслуживает пристального внимания, особенно если эти программы могут использоваться только последовательно и при вмешательстве всех остальных действий мультипрограммирования.

Таким образом, наиболее важной целью проектирования модуля приема в операционной системе является организация программ трансляции запросов на работу во вторичные коды, используемые для мультипрограммирования.

В процессе выполнения прикладных программ пользователя выходные данные выдаются на устройствах с низким быстродействием, например, таких, как АЦПУ, устройство для перфорации перфокарт и т. д.

При наличии мультипрограммного режима обычно такие данные, которые выработала прикладная программа, не попадают непосредственно на периферийное устройство, так как оно может быть занято выдачей результатов другой программы, которая также обрабатывается операционной системой.

Обычно такие выходные данные посылаются в большую память с произвольным доступом, которая используется как буфер для промежуточного хранения данных перед выдачей их на периферийном устройстве с низким быстродействием.

Второй функцией ввода является считывание данных обратно в вычислительную систему так, чтобы они могли попасть на периферийное устройство вывода.

Программы операционной системы, которые предназначены выполнять функции ввода-вывода данных, в дальнейшем будем называть модулем ввода-вывода операционной системы.

Из-за большого несоответствия между способностью периферийных устройств выдавать выходные данные и способностью системы формировать выходные данные для выдачи, весьма вероятно, что загруженная работой система никогда на самом деле не сможет выдать эти данные. С другой стороны, что может сильнее расстроить пользователя, который знает о том, что его работа завершена, ждет результатов, но видит, что печатающее устройство простаивает или работает мучительно медленно. Следовательно, необходимо проектировать систему не только так, чтобы сделать более быстрым выполнение модуля вывода, но и подсоединить этот модуль к системе так, чтобы обеспечить максимальную эффективность работы устройств.

### **Операционная система (первое определение)**

Мы рассматриваем операционную систему как набор системных программ, предназначенных выполнять три основные функции — принятия, выполнения и представления выходных данных, которые только что были описаны.

В последующих главах мы будем придерживаться той точки зрения, что прикладные программы, так же как и программы операционной системы, можно представлять как средства системы и что пользователя в системе представляют не его программы, а скорее его запрос, т. е. логическая единица, которая динамически устанавливается для пользователя в течение прохождения его работы через систему, включая выполнение его прикладных программ. *Запрос* — это та логическая единица, которой манипулирует операционная система. Запросы, представляющие пользователя, проходят в модуль ввода-вывода операционной системы и через нее в исполнительную часть, с этого момента они связаны с прикладными программами, которые загружались во время их приема.

Исполнительная часть ОС затем манипулирует всеми параллельными запросами в системе с тем, чтобы обеспечить динамическое распределение ЦП и средств ввода-вывода.

По мере того, как по запросу пользователя продолжается выполнение прикладных программ, по этому же запросу может формироваться выходная информация, предназначенная для устройства вывода. Эта выходная информация с помощью модуля вывода посылается на устройство с произвольным доступом и находится там до тех пор, пока устройство вывода не освободится. После этого можно выдать информацию и по окончании сигнализировать о том, что запрос пользователя завершил свое прохождение через модуль вывода операционной системы и, таким образом, прошел через всю систему. Мы можем, следовательно, представлять себе выполнение работы пользователя как прохождение соответствующего запроса через все части операционной системы.

Следует заметить, что в предшествующем описании операционной системы так называемая исполнительная часть рассматривалась как одна из нескольких программ, составляющих операционную систему, и кроме того, как часть, выполняющая специфические функции операционной системы, которые описывались ранее.

Таким образом, исполнительная часть относится к операционной системе, и настоящая работа не противоречит этому факту.

## Логическое разделение программ с помощью аппаратуры

В предыдущих параграфах мы рассматривали определенные функции операционной системы, которые она предназначена выполнять. Затем мы определили операционную систему как набор программ, составляющих модуль ввода, исполнительную часть и модуль вывода. Это предполагает, что все программы вычислительной системы делятся на две категории — категорию программ, относящихся к набору программ, составляющих операционную систему, и категорию, не относящуюся к ней. Последнюю категорию программ мы будем называть *прикладной*.

Рассмотрим выполнение прикладной программы. Работа, связанная с выполнением программы, пропускается через блок приема, в результате чего распределяется оперативная память, и программа загружается в нее; после этого исполнительный блок операционной системы динамически выделяет ЦП этой программе на определенные интервалы времени.

Процесс выполнения будет включать в себя выполнение инструкций прикладной программы, которые лежат за пределами набора программ, составляющих операционную систему. С другой стороны, так как функция исполнительного блока заключается и в том, чтобы распределять устройства ввода-вывода между прикладными программами, очевидно, необходимо, чтобы прикладная программа проходила обработку посредством исполнительных программ с тем, чтобы исполнительный блок выполнял свою функцию.

На языке запросов мы будем говорить, что запрос, представляющий прикладную программу, попадает в операционную систему, где с ним манипулирует запрос, представляющий исполнительные программы ОС.

Существует физическая характеристика всех современных вычислительных систем, которая имеет прямое соответствие со смысловым разделением прикладных программ и программ операционной системы. Она известна как *режим привилегированных команд* (Privileged instruction mode).

В случае, если система обладает этой возможностью, говорят, что она работает в одном или двух режимах.

Существует режим прикладных программ и режим операционной системы. Текущий режим системы задается аппаратным переключателем, который может быть переключен в то или иное состояние с помощью системы команд.

*Привилегированными командами* назовем те команды, которые могут выполняться только тогда, когда вычислительная система работает в режиме операционной системы.

Примером может служить набор команд, который используется для помещения файлов в периферийные устройства, и команды передачи данных между этими устройствами и оперативной памятью. Таким образом, для того, чтобы действительно выполнить операцию ввода-вывода, необходимо, чтобы ЦП перешел от выполнения команд прикладной программы к выполнению команд операционной системы. В частности, он должен перейти к выполнению этих команд при работе исполнительного блока ОС, который инициирует и работу с требуемыми периферийными устройствами.

## Прерывания

Программа, составляемая для ЭВМ, состоит из команд, которые выполняют определенные операции по проверке данных и манипуляции ими, а также других команд, осуществляющих условную или безусловную передачу управления на соответствующие точки этой программы. Последовательность выполнения команд является в этом случае динамической. Любое нарушение такой динамической последовательности, в результате которого управление передается на команды программы, отличной от той, которая обрабатывается до момента этого нарушения, называется *прерыванием*.

Механизм прерывания для разных машин может иметь свои отличительные особенности, но в основе его лежит один и тот же логический принцип. В момент прерывания режим вычислительной системы изменяется, и она начинает работать в режиме операционной системы. Адрес слова прерываемой программы, на котором произошло прерывание, запоминается, а следую-

шая команда, которая должна выполняться, берется из фиксированного слова оперативной памяти. Прерывание буквально заставляет вычислительную систему перейти от прикладной программы к операционной системе. Кроме того, место прикладной программы, на котором происходит прерывание, запоминается, и после прерывания обработка прерванной программы начинается с этого же места. И, наконец, следующая команда берется из фиксированного слова оперативной памяти так, что все последующие логические действия вычислительной системы определяются теми командами, которые размещаются в массиве слов, начиная с вышеуказанной. Команда, выбранная из фиксированной ячейки, будет первой командой операционной системы, которая выполняется после прерывания и изменения режима.

Существуют три категории прерываний, каждая из которых будет описана ниже. Прерывания, относящиеся к любой из этих категорий, следует отличать также и по причине, которая их вызвала.

Следовательно, логические функции, которые должны будут выполняться при прерывании за счет выполнения программ операционной системы, обычно будут начинаться с обработки прерывания, целью которой является определение категории и распознавание каждого отдельного прерывания.

Как уже говорилось выше, прерывание можно разделить на три категории: категорию *сигнальных прерываний*, категорию прерываний *вынужденного входа* и категорию *генерируемых прерываний*.

Сигнальные прерывания возникают за счет работы внешних переключателей, которые используются для того, чтобы сообщать операционной системе о состоянии внешних устройств.

Так, например, на операторском пульте обычно предусмотрена кнопка сигнального прерывания, которая используется оператором в авостной ситуации. Прерывания, относящиеся к категории сигнальных, могут быть вызваны различными причинами и поэтому отличаются друг от друга. Например, прерывание, возникшее за счет нажатия кнопки на пульте, не будет эквивалентно прерыванию, скажем, за счет установки ленты на магнитофон.

Прерывания категории вынужденного входа возникает в результате выполнения привилегированных команд, которые, могут выполняться только в режиме операционной системы, а не в прикладном режиме.

Например, выполнение макрокоманды READ (читай) в прикладной программе вызывает переход на операционную систему, в которой программы ввода-вывода исполнительного блока будут осуществлять требуемое управление устройствами и передачей данных. Такие прерывания должны быть идентифицированы с тем, чтобы можно было различить, например, прерывание для макрокоманды READ и прерывание для макрокоманды WRITE (пиши).

Прерывания могут происходить и через некоторый промежуток времени после того, как они сгенерированы. Такие прерывания относятся к категории генерируемых. Примером генерируемого прерывания может служить прерывание ввода-вывода после того, как работа периферийного устройства завершена. Идентификация прерываний этого класса необходима для того, чтобы определить, какое устройство завершило свою работу и тем самым вызвало прерывание.

Из сказанного выше ясно, что введение понятия привилегированных команд в современных машинах требует, чтобы прикладные программы и программы операционной системы логически отделялись. Это достигается за счет реализации двух режимов работы вычислительной системы, при которых пользователь получает доступ к операционной системе только через прерывания вынужденного входа. В последующих главах мы рассмотрим систему обозначений, которая приспособлена для отделения прикладных программ от программ операционной системы.

### **Защита и распределение памяти**

Следует сказать также несколько слов относительно защиты прикладных программ друг от друга. Вообще пользователь как бы не подозревает о прикладных программах других пользователей, обрабатываемых параллельно с его программой. В частности же, такое совместное пользование памятью приводит к тому, что ячейки памяти, приписанные каждой программе, пол-

ностью должны контролироваться операционной системой. Операционная система создает архив для информации, касающейся распределения памяти для параллельно решаемых программ, организуя этот архив по принципу так называемой *карты памяти*.

Так как последовательность запланированных ячеек постоянно приписывается программам, то операционная система, зная о неприписанных или свободных ячейках, работает как распределитель с тем, чтобы дать возможность новым прикладным программам войти в обработку.

Некоторые вычислительные машины имеют аппаратную возможность *защиты памяти*. Кратко такую возможность можно определить как механизм «удерживания» программы в отведенной ей области памяти. Так как распределение памяти должно обязательно входить в число функций операционной системы, то и защита памяти также, очевидно, должна реализоваться операционной системой. Таким образом, защита памяти представляет собой некое средство для изоляции или отделения прикладных программ друг от друга, а также для отделения их от программ операционной системы.



## ГЛАВА 3

### НЕКОТОРЫЕ ОБЩИЕ СООБРАЖЕНИЯ

Рассмотрим вычислительную систему, имеющую значительное количество периферийных устройств, доступ к которым осуществляется через ряд каналов ввода-вывода.

Предположим также, что эта система имеет весьма большую оперативную память и очень быстродействующий центральный процессор.

Ради примера возьмем систему, обладающую вполне определенным количественным составом, т. е. предположим, что в нее входит два набора магнитофонов, на которые можно поставить восемь лент, причем каждый набор подключен к системе через канал ввода-вывода. Кроме того, предположим, что система обладает устройством памяти с произвольным доступом (МД, МБ), АЦПУ и устройством чтения перфокарт, каждое из которых подключено к системе через свой собственный канал.

Будем считать, что память содержит 96 К ячеек и после отведения в ней места для операционной системы свободными остаются 90 К ячеек. И, наконец, предположим, что среднее время выполнения операций для ЦП составляет 2 *мксек*, причем время выборки/обращения к оперативной памяти достаточно мало и может хорошо согласовываться с указанным быстродействием ЦП.

Во-первых, очевидно, что средний пользователь такой системы не в состоянии будет использовать всю ее мощность, так как только огромная программа может

работать с 16 магнитофонами и использовать все возможное время обращения к устройству памяти с произвольным доступом.

Во-вторых, вероятность того, что программа займет более половины всего объема оперативной памяти ( $1/2 \times 90$  К), весьма мала, не говоря уже о полном ее объеме. Скорее всего, объем памяти, занимаемой программой, не составит и ее половины ( $1/2 \times 90$  К).

И, наконец, весьма вероятно, что программа, использующая обширный набор периферийных устройств, будет характеризоваться большой частотой команд, которые будут требовать ввода-вывода для этих устройств.

Следовательно, можно предполагать, что время вычислений между операциями ввода-вывода будет невелико. Из этого, в свою очередь, вытекает, что при среднем времени выполнения операций, равном 2 мксек, и максимальном быстродействии самых современных периферийных устройств центральный процессор будет простаивать в течение продолжительного промежутка времени. Даже в более простой ситуации, когда все средства вычислительной системы полностью заняты обработкой только одной программы и при этом используется лишь несколько периферийных устройств, мы вновь столкнемся с простоями.

Такое утверждение легко распространяется и на неиспользуемую память вычислительной системы, которая бездействует в течение периода времени, когда вся вычислительная система обрабатывает одну программу.

Даже для тех программ, которые способны достигать хорошо сбалансированной буферизации, но не используют всех периферийных устройств, пропускная способность простаивающих устройств равна нулю.

Заметим, что даже и в том случае, если программа использует все периферийные устройства или большую часть из них, вопрос о достижении высокого уровня балансирования ввода-вывода остается весьма спорным из-за громадного несоответствия между быстродействием периферийных устройств и ЦП.

Следовательно, целью мультипрограммирования является организация распределения средств вычисли-

тельной системы, т. е. периферийных устройств, ЦП и оперативной памяти по нескольким программам.

Эти программы одновременно находятся в системе и при этом одновременно используют память и набор периферийных устройств. ЦП используется последовательно.

### **Квантование времени**

*Мультипрограммированием* называется метод, реализованный программно, позволяющий ввести в вычислительную машину более чем одну программу для последующей параллельной обработки. Основной задачей мультипрограммирования до недавнего времени было осуществление доступа к вычислительной системе более чем одного пользователя, в один и тот же промежуток времени. Вторая задача заключалась в том, чтобы повысить пропускную способность вычислительной системы.

Одним из методов, позволяющих осуществить такой мультидоступ к вычислительной системе, является мультипрограммирование с *квантованием времени*.

Каждой готовой к выполнению программе в системе на соответствующий промежуток времени отдаются все средства вычислительной системы. Такой промежуток времени называется квантом времени. По окончании его все средства системы передаются от текущей программы другой программе.

Следовательно, если следующая программа, которой должно быть отдано управление средствами системы, постоянно находится в оперативной памяти, то процедура передачи управления и квантование времени не вызывает осложнений. С другой стороны, если программа постоянно находится на периферийном устройстве, таком, например, как память с произвольным доступом, то для необходимого функционирования этой программы требуется, чтобы операционная система освободила место в оперативной памяти (обычно это делается за счет сброса другой программы) и вызвала ее на это место.

При таком подходе к мультипрограммированию задачей операционной системы является контроль за выполнением выбранной программы, находящейся в опе-

ративной памяти, в течение заранее установленного промежутка времени.

Когда квант времени, выделенный программе, истекает, операционная система осуществляет ее прерывание, запоминает всю необходимую информацию о состоянии прерываемой программы для ее последующей выборки и после этого выбирает программу, которая будет обрабатываться следующей.

При таком методе мультипрограммирования обработка программ остается по существу последовательной, с той лишь разницей, что процессор обрабатывает последовательно части программ, а не программы в целом.

Общее время нахождения программы в памяти увеличивается, пропускная способность не улучшается, и единственным подлинным достоинством такого варианта мультипрограммирования является возможность установления связи с обрабатываемой программой. Такая возможность может быть реализована пользователями, работающими параллельно на выносных пультах. Каждый из пользователей может в этом случае обратиться в тех или иных целях к собственной программе.

Мультипрограммирование с квантованием времени все же улучшает эффективность системы по одному из ее параметров, а именно — степени использования оперативной памяти. Это повышение эффективности достигается за счет того, что для достаточно большой оперативной памяти и при соответствующей сортировке программ пользователя по величине несколько из них могут параллельно находиться в оперативной памяти.

С другой стороны, мультипрограммирование с квантованием времени ничего не дает в смысле повышения эффективности использования ЦП или периферийных устройств системы. Это связано с тем, что в системе всего один центральный процессор, каждый раз он обрабатывает всего одну программу. Поэтому обработка нескольких программ ведется им строго последовательно. То же самое можно сказать и о периферийных устройствах. Хотя они и закрепляются за программами параллельно, одновременно обслуживать разные программы такие устройства не могут.

Это обусловлено тем, что при реализации метода квантования времени управление устройствами отдается каждый раз только одной из программ, ожидающих своей очереди. Следовательно, когда программа ставится на обработку, одновременно могут работать только те устройства, которые закреплены за данной программой. Так как другие программы в это время не выполняются, связанные с ними периферийные устройства работать не будут.

Из сказанного выше можно сделать вывод, что мультипрограммирование с квантованием времени дает весьма мало в смысле повышения пропускной способности вычислительной системы, обеспечивая лишь небольшой выигрыш во времени решения с точки зрения пользователя.

Фактически этот тип мультипрограммирования действительно эффективен лишь тогда, когда он применяется в так называемом режиме разделения времени. Это связано с тем, что в этом режиме существует реальная необходимость откладывать решение задачи до тех пор, пока не придет запрос с удаленного терминала пользователя. Если большое количество пользователей параллельно должно иметь доступ к системе, задача мультипрограммирования с квантованием времени заключается в том, чтобы система последовательно реагировала на запросы (приказы, сигналы, директивы) пользователей с терминалов через интервал времени, равный кванту времени.

Из сказанного выше видно, что нужно искать другие методы мультипрограммирования, которые увеличили бы эффективность и ЦП и периферийных устройств, а также оперативной памяти.

Для того, чтобы повысить эффективность использования ЦП, нужно организовать его работу так, чтобы он обрабатывал команды всех программ, за исключением тех программ, выполнение которых приостанавливается за счет операций ввода-вывода.

С другой стороны, цели, к которым стремятся с помощью мультипрограммирования, относительно периферийных устройств более обширны и не ограничиваются вышесказанным.

Задача сводится к тому, чтобы максимально распараллелить работу периферийных устройств системы

и при этом увеличить пропускную способность каждого устройства.

В дальнейшем мы увидим, что эти две цели не всегда совместимы.

### **Мультипрограммирование запросов на ввод-вывод**

Другим подходом к мультипрограммированию с квантованием времени является метод, который в дальнейшем мы будем называть *мультипрограммированием запросов на ввод-вывод*.

Говоря попросту, этот метод использует кванты времени переменной длины, которая определяется интервалом времени между прерываниями после окончания ввода-вывода.

Как только периферийное устройство завершило ввод-вывод, операционная система немедленно иницирует следующий ввод-вывод, при условии, если такая необходимость существует в данный момент времени. Таким образом, операции ввода-вывода как бы управляют системой в том смысле, что при их появлении осуществляется выбор новой программы, которая будет обрабатываться. Из этого вытекает, что система ориентируется на появление запросов на ввод-вывод, распределяет и перераспределяет ЦП на основе этих запросов и связанных с ними прерываний.

Следует заметить, что ситуация, возникающая при использовании такого типа мультипрограммирования, несколько отлична от той, которая имела место в случае мультипрограммирования с квантованием времени, где основной целью было распределение ЦП между программами.

Одним из важных преимуществ мультипрограммирования запросов на ввод-вывод является то, что оно обеспечивает более единообразное распределение ЦП по программам и осуществляет выбор следующей программы сразу по окончании ввода-вывода так, что существует по крайней мере одна программа, которую может обрабатывать ЦП.

Более важным, однако, является то, что этот тип мультипрограммирования создает благоприятную обстановку для более быстрого распределения ЦП по

всем программам и в связи с этим расширяет возможности инициирования различных операций по вводу-выводу.

Теперь должно быть ясно, что фактическим различием между мультипрограммированием с квантованием времени и мультипрограммированием запросов на ввод-вывод является вопрос организации управления.

При реализации мультипрограммирования с квантованием времени программе пользователя отдается управление всеми средствами системы, которые используются в ней. Смена обрабатываемой программы происходит только тогда, когда время, равное кванту времени, истекло или если в ней самой задано условие того, что она должна попасть во временный буфер и ожидать, пока не закончится передача данных.

В последнем случае необходимо, чтобы обрабатываемая программа передала управление центральным процессором операционной системе.

Ситуация, возникающая при реализации мультипрограммирования запросов на ввод-вывод, как раз противоположна описанной выше. В этом случае ОС постоянно управляет периферийными устройствами, отдавая программе пользователя управление только центральным процессором. ОС следит за вводом-выводом, и когда выполнение программы должно приостановиться из-за передачи данных, операционная система берет управление центральным процессором на себя. Именно таким способом можно достичь высокого уровня распараллеливания периферийных устройств.

Однако, как отмечалось ранее, одной из целей, которые мы преследуем в отношении периферийных устройств, является повышение пропускной способности каждого устройства в отдельности. Указывалось также, что увеличение пропускной способности и распараллеливание работы устройств, как правило, одновременно не достигается, т. е. наблюдается некая несовместимость этих тенденций. В связи с этим мы рассмотрим далее два различных типа мультипрограммирования запросов на ввод-вывод, каждый из которых обеспечивает либо увеличение пропускной способности, либо распараллеливание при возможном ухудшении другой характеристики.

## Небуферируемый запрос на ввод-вывод

Простейшей и наиболее прямолинейной считается такая вычислительная система, в которой все операции ввода-вывода программы пользователя выполняются без буферизации. Это означает, что если в программе встретились команды ввода-вывода, выполнение которых вызывает передачу данных между периферийным устройством и оперативной памятью, то ЦП не будет обрабатывать программу дальше до тех пор, пока эта передача данных не завершилась. По окончании передачи данных происходит прерывание по вводу-выводу.

Если программа иницирует ввод-вывод, то она необходимо должна быть текущей обрабатываемой программой, и для того, чтобы выполнить этот ввод-вывод, она должна передать управление операционной системе. Если периферийное устройство, закрепленное за данной программой, свободно, то требуемая функция ввода-вывода выполняется сразу, если же это устройство занято, программа ставится в очередь на данное устройство. Таким образом, дальнейшее выполнение программы откладывается до тех пор, пока не завершится передача данных, и тогда только будет исполнен запрос на ввод-вывод.

Следующим шагом операционной системы является выбор новой программы для обработки и переход к ее выполнению.

Таким образом, каждый запрос на ввод-вывод вызывает перераспределение ЦП, т. е. программы, обрабатываемые ЦП, меняются. Ясно, что весь набор периферийных устройств будет активизироваться и работать с высокой степенью распараллеливания благодаря тому, что ЦП распределяется между всеми программами, которые обрабатываются системой. С другой стороны, ясно, что при таком режиме работы пропускная способность всего набора устройств системы в пересчете на одно устройство может стать и минимальной.

Вероятнее всего, что с такой же пропускной способностью будут работать устройства с последовательным принципом действия, такие, как магнитофоны, перфорирующие устройства и т. д. Это может иметь место и для отдельных устройств памяти с произвольным доступом. Что касается магнитофона, то он обычно при-



писывается одной и только одной программе. Предположим, что этот магнитофон используется, скажем, для ввода. Это означает, что каждый вызов следующей записи, который освобождает вводный блок и инициирует передачу данных, будет задерживать дальнейшее выполнение программы до тех пор, пока обмен с лентой не завершится. Вследствие этого пройдет значительное время, пока программа вновь не будет выбрана в качестве текущей обрабатываемой. В этом случае будут возникать также и сопутствующие потери, если старто-стопное время для этого магнитофона соизмеримо с полным временем обмена.

Рассмотрим теперь устройство памяти с произвольным доступом, такое, например, как диск, которое имеет единственное перемещаемое устройство для выборки данных. При последовательных считываниях из файла, находящегося на диске, та программа, которая инициирует эти считывания, может весьма успешно воспользоваться текущим положением устройства для выборки данных в целях минимизации времени поиска и увеличения пропускной способности диска относительно отдельной программы.

С другой стороны, если несколько программ используют этот диск параллельно в условиях небуферируемого запроса на ввод-вывод, его пропускная способность может стать минимальной из-за обилия и различного местоположения просматриваемых участков диска. Ясно, однако, что такая ситуация исключена при использовании дисков, имеющих по считывающей головке на каждый тракт, или барабанов с фиксированными головками. Здесь время выборки по существу является временем простоя задачи, и с ним надо мириться во всех случаях жизни, так что переключение с одной программы на другую и последовательное переключение с одной области доступа на другую фактически не влияют на пропускную способность таких устройств.

### **Буферируемый запрос на ввод-вывод**

Под *буферизацией* понимается метод, который позволяет программе пользователя инициировать ввод-вывод и затем продолжать ее выполнение параллельно

с выполнением передачи данных. В условиях мультипрограммирования без буферизации происходит нечто подобное, только в тот момент, когда осуществляется передача данных для одной программы, а ЦП обрабатывает некоторую другую программу и это продолжается до тех пор, пока в системе находятся по крайней мере две программы.

Однако сейчас нас интересует случай, когда операции ввода-вывода и обработка на ЦП выполняются параллельно для одной и той же программы. Такой тип мультипрограммирования, в котором реализована буферизация, обладает, вероятно, двойным преимуществом по сравнению с предыдущими. Во-первых, значительно уменьшается вероятность такой ситуации, когда ЦП свободен, но нет программы, которая могла бы им обрабатываться. Во-вторых, появляется возможность буферизации данных необходимых для повышения пропускной способности отдельных периферийных устройств. В частности, это относится к дискам с подвижным устройством выборки данных, некоторым устройствам ввода-вывода данных, таким, например, как быстродействующие АЦПУ, и другим устройствам с последовательным принципом действия.

С другой стороны, существует потенциальная сопутствующая трудность, связанная с буферизацией. Ранее уже указывалось, что основной целью операционной системы являлись повышение пропускной способности системы в целом за счет улучшения пропускной способности каждого периферийного устройства, а также повышение параллельности использования этих периферийных устройств. Мультипрограммирование без буферизации имеет тенденцию к повышению параллельности использования периферийных устройств и в то же самое время снижает пропускную способность отдельных устройств.

В данном случае программа пользователя, выполняемая в режиме буферизации, одновременно осуществляет управление не только ЦП, но и целым рядом периферийных устройств, закрепленных за ней. Следовательно, максимальный уровень распараллеливания периферийных устройств определяется распараллеливанием, которое может обеспечить эта программа для приписанных ей устройств. Отсюда видно, что при ме-

тоде мультипрограммирования с буферизацией улучшается пропускная способность за счет снижения уровня распараллеливания работы периферийных устройств.

### **Замечания**

Из сказанного выше трудно сделать вывод о том, какой тип мультипрограммирования (с буферизацией или без нее) предпочтительней. Очевидно, что выбор типа, в основном, зависит от целей, которые он преследует. Например, если целью ставится повышение пропускной способности системы, то режим без буферизации, возможно, и позволил бы решить эту проблему. Однако, если при этом необходимо обеспечить минимальное время решения, метод мультипрограммирования с буферизацией был бы более плодотворным.

Следует, однако, понимать, что проблема выбора решается не так просто, как это могло бы показаться при прочтении предыдущего абзаца. Например, при попытке увеличить пропускную способность системы (если допустимо некоторое увеличение индивидуального времени решения) за счет выполнения ввода-вывода без буферизации для каждой программы может случиться так, что пропускная способность в пересчете на одно устройство недопустимо уменьшится.

Например, при попытке увеличить пропускную способность системы за счет небуферуемого выполнения ввода-вывода для каждой программы пропускная способность в пересчете на одно устройство может снизиться настолько, что все усилия, направленные на увеличение общей пропускной способности, будут сведены на нет.

То же самое можно сказать и в отношении времени решения. Если, например, попытаться уменьшить время решения для выбранного набора программ в системе, то время решения для всех остальных программ увеличится. Кроме того, при выборе возникают вопросы, связанные с уровнем буферизации программ и стоимостью эксплуатации самого механизма буферизации.

Очевидно, что выбор типа мультипрограммирования не так-то прост и в значительной степени зависит

от специфики вычислительной системы и ее эксплуатации.

Поскольку цель данной работы как раз и заключается в освещении, выделении и разработке методики построения и анализа операционных систем, рассмотрение вопросов, касающихся буферизации, стоимости, эксплуатации и т. д., мы перенесем в разделы, специально для этого предназначенные. Так, в следующей главе мы рассматриваем наиболее простой, с точки зрения операционной системы, случай мультипрограммирования — мультипрограммирование небуферируемых запросов на ввод-вывод.

В последующих главах мы будем более детально рассматривать сущность буферизации и затем несколько подробнее — операционные системы, в которых такая буферизация предусматривается.

## ГЛАВА 4

### **ИСПОЛНИТЕЛЬ НЕБУФЕРИРУЕМОГО ЗАПРОСА НА ВВОД-ВЫВОД**

В главе 3 уже излагались мотивы, побуждавшие нас рассмотреть тип мультипрограммирования (или такую ОС), при реализации которого все операции ввода-вывода для программ пользователя выполняются только без буферизации.

С помощью варианта мультипрограммирования без буферизации (типа ОС) можно повысить уровень распараллеливания периферийных устройств системы. Характерно, что мультипрограммный исполнитель (MPX-multiprogram executive) полностью определяется приведенным выше условием, т.е. для всех программ ввод-вывод осуществляется без буферизации. В этой главе мы приведем пример разработки MPX для мультипрограммирования небуферизируемого запроса на ввод-вывод, на основе современных методов. В главе 5 мы снова возвратимся к разработке такого MPX, но уже на уровне анализа запросов.

Когда все это будет выполнено, мы перейдем (в главе 7) к анализу простой, но удивительно мощной системы мультипрограммирования, причем MPX для нее будет тем самым, который мы рассмотрим в предыдущих главах.

#### **Общее описание**

Ядром операционной системы является мультипрограммный исполнитель (MPX), и именно он определяет эффективность системы в целом. В дальнейшем при-

ладные программы мы будем называть *рабочими программами*. Это делается для того, чтобы в дальнейшем можно было четко различать прикладные программы и программы операционной системы. Набор прикладных программ, которыми управляет исполнитель, будем называть *мультипрограммным набором*.

Основной задачей исполнителя, разрабатываемого в этой главе, является выборочное управление параллельным выполнением рабочих программ из мультипрограммного набора. При этом исполнитель должен осуществлять управление операциями ввода-вывода для рабочих программ так, чтобы достичь максимального распараллеливания работы периферийных устройств.

Выбор программ для выполнения будет осуществляться на основе системы приоритетов.

Очевидно, в первую очередь нужно рассмотреть, как сохраняется информация о состоянии каждой программы из набора в любой заданный момент времени относительно ее операций ввода-вывода. В связи с этим мы начнем наше рассмотрение с обзора состояний, в которых может находиться *небуферируемая* рабочая программа, и причин, эти состояния обуславливающих.

### Состояния программы

Во-первых, предположим, что текущая рабочая программа, которая обрабатывается ЦП, обратилась к устройству ввода-вывода, подключенному к одному из каналов. Допустим, что при этом либо устройство, либо и канал и устройство заняты. Говорят, что при такой ситуации программа находится в состоянии *ожидания* (инициирования) *начала* операций ввода-вывода. Так как программа — *небуферируемая*, в этот момент она не может выполняться, и нельзя продолжить ее обработку параллельно с выполнением операций ввода-вывода.

Во-вторых, предположим, что текущая рабочая программа обратилась к свободному устройству, причем канал также не занят. В этом случае иницируется функционирование ввода-вывода, и программа находится в состоянии *ожидания конца* ввода-вывода.

Понятно, что продолжить выполнение небуферируемой программы нельзя.

В-третьих, программа, не ожидающая ни начала ввода-вывода, ни его конца, будет находиться в состоянии *готовности* к обработке. В дополнение к этому следует заметить, что текущая выполняемая программа также может находиться в этом состоянии.

Обзор приведенных выше трех состояний показывает, что все они взаимосвязаны и взаимоисключающи для каждой программы из мультипрограммного набора. Т. е. каждая программа из мультипрограммного набора может находиться только в одном из приведенных ниже состояний:

- 1) ожидания начала ввода-вывода;
- 2) ожидания завершения ввода-вывода;
- 3) готовности к обработке или дальнейшему выполнению.

Ясно, что, если программа находится в состоянии 3, то в какой-то момент она будет выбрана в качестве текущей обрабатываемой. В процессе выполнения этой программы может потребоваться обмен с периферийным устройством. Если это происходит — программа из состояния 3 переходит в состояние 1 или 2, в зависимости от сложившейся ситуации (т. е. в зависимости от того, были ли к данному моменту времени заняты канал и требуемое устройство или же они были свободны).

Таким образом, если обмен (передача данных) не может быть осуществлен в данный момент времени, программа перейдет в состояние 1 и будет находиться в нем до тех пор, пока необходимые для обмена средства (канал, устройство) не освободятся. После этого будет начат обмен (передача данных) для данной программы, и она перейдет в состояние 2. Если же средства для обмена были свободны тогда, когда программа находилась в состоянии 3, то обмен начинается сразу же, и программа из состояния 3 переходит прямо в состояние 2.

Однако в любом случае программа перейдет в состояние 2 и будет находиться в нем до тех пор, пока иницируемый ею обмен не закончится, после чего она перейдет в состояние 3. Это перемещение программы из состояния в состояние иллюстрируется показанной

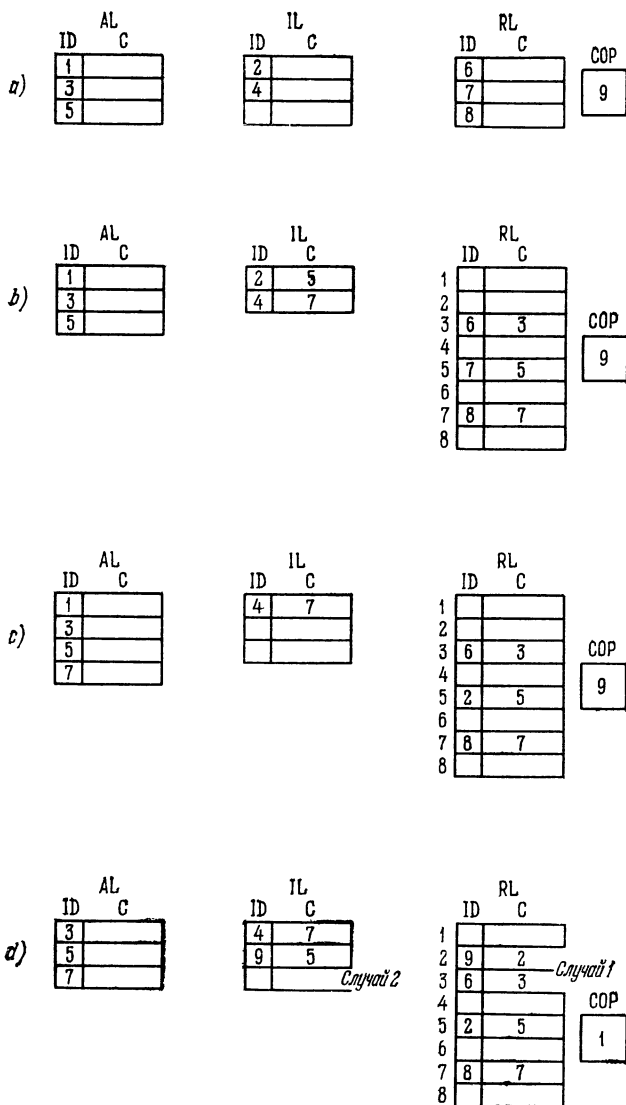
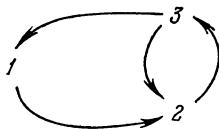


Рис. 1. Организация контроля состояний программ для исполнителя задач.



ниже схемой переходов:



Для того, чтобы исполнительная часть ОС МРХ могла фиксировать состояние каждой программы из мультипрограммного набора (смеси), нужны три списка (таблицы), к которым мы иногда будем обращаться: *список готовности* AL (available list), *список ожидания начала ввода-вывода* IL (initiation list) и *список ссылок (обращений)* RL (reference list).

На рис. 1 программа, стоящая в столбце ID (идентификатор программы списка AL), готова к выполнению (состояние 3). Программа, занесенная в список IL, ожидает инициирования ввода-вывода (состояние 1), и, наконец, программа, попавшая в список RL, ожидает завершения ввода-вывода, который уже инициирован и выполняется (состояние 2).

При таком подходе мультипрограммный исполнитель, очевидно, должен служить как некое связующее звено между программами из мультипрограммного набора и блоком ввода-вывода операционной системы, т. е. каждый запрос на ввод-вывод должен адресоваться непосредственно мультипрограммному исполнителю. Исполнитель, в свою очередь, при наличии свободных устройств и каналов, передает эти запросы в блок ввода-вывода операционной системы согласно их приоритетам. Прерывания, завершающие ввод-вывод, будут проходить через блок ввода-вывода ОС. В силу того, что МРХ выполняет такую роль в системе, рассмотрим некоторые возможности и методы контроля состояний программ. Мы начнем с примера, иллюстрирующего условия, при которых происходит переход из одного состояния в другое.

На рис. 1, а даны четыре таблицы, демонстрирующие состояния программ 1—9. Так, программы 1, 3 и 5 готовы к выполнению (см. список AL). Программы 2 и 4 ожидают инициирования ввода-вывода (см. список IL); программы 6, 7 и 8 ждут окончания ввода-вывода (см. список RL). Девятая программа обрабаты-

вается в данный момент (текущая обрабатываемая), см. СОР.

Теперь возникает вопрос, как исполнитель узнает о том, что для программ 2 и 4 в данный момент ввод-вывод инициироваться не может (несмотря на то, что исполнитель уже получил соответствующие запросы), и как ему станет известно о завершении ввода-вывода для программ 6, 7 и 8? Видно, что таблицы на рис. 1, *a* дают явно недостаточную информацию для ответа на поставленные вопросы. Однако некоторая модификация списка RL позволила бы это сделать достаточно просто. Предположим, что вычислительная система располагает восемью каналами. Тогда построим список RL так, чтобы он состоял из восьми слов (строк), причем каждое слово соответствовало бы одному из каналов (рис. 1, *b*). В дальнейшем такие слова будем называть словами состояния каналов. При требовании ввода-вывода запросы, посылаемые исполнителю, вызывают заполнение столбца ID слов состояния каналов идентификаторами тех программ, которые нуждаются во вводе-выводе. Причем заполняются лишь те слова, содержимое которых в данный момент равно нулю.

Присутствие ненулевой информации в столбце ID слова состояния канала свидетельствует теперь о том, что канал занят.

В нашем случае программы 6, 7 и 8 занимают каналы 3, 5 и 7 соответственно. Если программы 2 и 4 потребуют ввода-вывода через каналы 5 и 7, то, очевидно, им придется ждать, так как эти каналы уже заняты 7 и 8 программами. MPX поместит идентификаторы программ 2 и 4 в столбец ID списка IL, кроме того, в столбец C будут занесены номера требуемых каналов.

Предположим, что в то время, пока обрабатывается программа 9, закончился ввод-вывод через канал 5, о чем свидетельствует завершающее ввод-вывод прерывание. Исполнитель в этом случае переместит ID для программы 7 из пятого слова RL в список готовности AL. После этого пятое слово RL будет обнулено. Затем исполнитель просмотрит список IL с тем, чтобы выяснить, какая программа должна использовать канал 5.

В нашем случае такой программой оказывается программа 2, и так как нужный канал освободился, ввод-

вывод может начаться. В силу этого ID для программы 2 перемещается из списка IL в пятое слово списка RL, и ввод-вывод начинается. После этого управление передается на текущую обрабатываемую программу 9, в состоянии, показанном на рис. 1, с.

Теперь предположим, что текущая обрабатываемая программа 9 запрашивает MPX на предмет выполнения ввода-вывода. Если для этого ввода-вывода требуется канал 2, который в нашем случае свободен, то ввод-вывод инициируется немедленно, а ID для девятой программы помещается во второе слово списка RL (рис. 1, d), случай 1). Если же для ввода-вывода программы 9 требуется пятый канал (занятый в данный момент программой 2), то ID девятой программы помещается в список IL (случай 2). Если же обработка программы 9 завершена полностью, ее ID удаляется вообще из всех списков и из AL, в частности. После этого исполнитель просматривает таблицу AL и выбирает одну из программ, находящихся в этом списке в качестве текущей обрабатываемой COP (Current Operating Program). Если COP стала программа 1, то система переходит в состояние, которое демонстрируется рис. 1, d (случай 1 или случай 2).

### Приоритет

В предыдущем примере мы рассмотрели, как MPX осуществляет контроль состояний программ из мультипрограммного набора (смеси), однако совсем не затрагивался вопрос контроля приоритетов, которыми могут обладать программы. Такой контроль можно осуществить, заменив два списка на очереди. Так, вместо таблицы готовности AL мультипрограммный исполнитель будет работать с *очередью готовности* AQ. В этой очереди будут содержаться ID программ, готовых к выполнению и размещенных в соответствии с системой приоритетов (т. е. по убыванию или по возрастанию приоритетов). В нашем случае ID располагаются в порядке возрастания приоритетов.

Список IL тоже меняется на очередь *ожидания ввода-вывода* IQ. Так же как и в случае AQ, ID заносятся в IQ в соответствии с приоритетами. Кроме того, в IQ содержатся номера каналов. На рис. 2 демонстрируется

структура очередей IL и IQ и списка ссылок RL, которые использует MPX при контроле состояний программ с учетом приоритетов. Рис. 2 отражает также

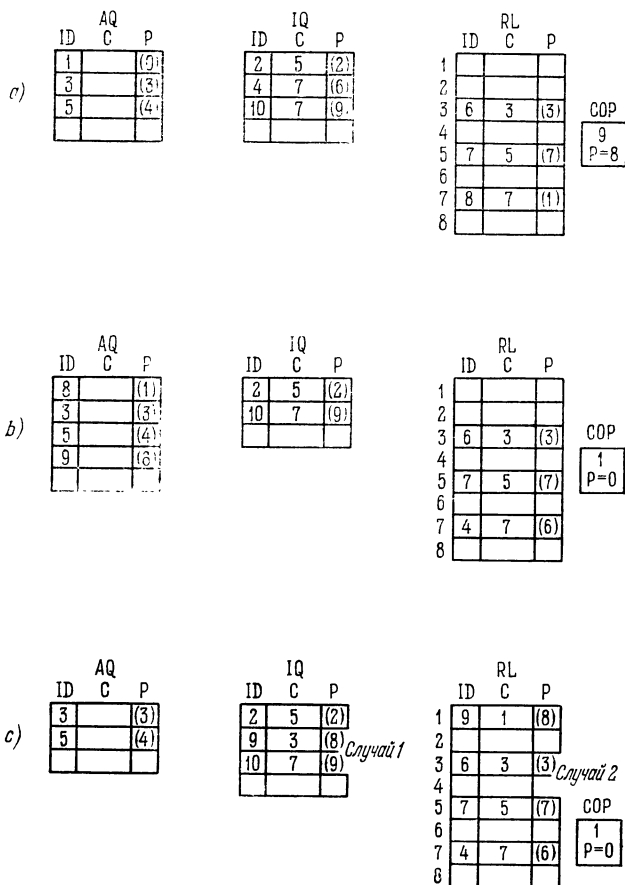


Рис. 2. Организация контроля состояний программ с учетом приоритетов.

процесс управления по приоритетам и учет состояний программ при выборе их для текущей обработки и при инициировании операций ввода-вывода. На рис. 2, а) дано некоторое исходное состояние списков и оче-

редей для каждого из ниже рассмотренных случаев. Приоритеты заключены в круглые скобки.

Предположим, что канал 7 освободился, о чем свидетельствовало прерывание, завершающее ввод-вывод. Это означает, что ввод-вывод для программы 8 закончился и что эта программа готова к выполнению. Тогда исполнитель переместит ID этой программы в AQ. Заметим при этом, что ID разместится в очереди готовности в соответствии с приоритетом программы 8. Седьмое слово RL обнуляется, чтобы указать на то, что канал освободился. Обнуление происходит сразу после того, как ID удаляется из RL. Таким образом, программа 8 переходит из состояния 2 в состояние 3.

После этого исполнитель просматривает очередь IQ и находит две программы, ждущие начала ввода-вывода и использующие один и тот же канал 7. В силу того, что значение приоритета программы 4 меньше, чем у программы 10, ввод-вывод будет инициироваться для нее в первую очередь. Как уже описывалось ранее, TD программы 4 перемещается в список ссылок RL и заносится в седьмое слово состояния канала (так как программа использует седьмой канал). Таким образом, программа 4 переходит из состояния 1 в состояние 2.

После этого исполнитель просматривает очередь готовности и обнаруживает, что программа 1, обладающая самым высоким приоритетом, готова к выполнению. Он прерывает выполнение программы 9 и выбирает в качестве текущей обрабатываемой программу 1. ID программы 9 в соответствии с ее приоритетом помещается в очередь AQ, так как теперь она находится в состоянии 3. Эту последовательность действий легко проследить, обратившись к рис. 2, *a* и *b*.

На рис. 2, *c* отражен случай, когда текущая обрабатываемая программа требует выполнения ввода-вывода. Исходным является состояние системы, показанное на рис. 2, *a*. Предположим, что для выполнения ввода-вывода требуется третий канал (см. средний столбец IQ). Так как он занят, исполнитель заносит ID программы 9, в соответствии с ее приоритетом, в очередь ожидания IQ. В силу того, что состояние текущей обрабатываемой программы фактически эквивалентно состоянию 3, можно сказать, что программа 9 переходит из состояния 3 в состояние 1. В качестве

текущей обрабатываемой программы выбирается программа 1, так как она обладает самым высоким приоритетом.

Теперь предположим, что программа 9 запрашивает канал 1 (исходным является состояние системы, показанное на рис. 2, а). Из списка ссылок видно, что первое слово состояния канала не содержит никакой информации и, следовательно, канал 1 свободен. В случае 2 ID программы 9 перемещается из COP (AQ) в первое слово RL, а соответствующее COP слово в AQ обнуляется. Иницируется ввод-вывод для программы 9. В качестве текущей обрабатываемой, как и в предыдущем случае, выбирается программа 1.

Очевидно, приведенных выше двух очередей и списка ссылок вполне достаточно для того, чтобы контролировать состояния небуферируемых рабочих программ, а также для того, чтобы выбирать программы для текущей обработки и инициировать ввод-вывод на основе системы приоритетов. Ниже мы подробно рассмотрим алгоритм контроля и функционирование мультипрограммного исполнителя. На рис. 3—6 даны блок-схемы MPX, которые иллюстрируют процесс управления запросами на ввод-вывод, прерываниями, завершением операций ввода-вывода и программ.

### **Запрос на ввод-вывод**

Обратившись к рис. 3, рассмотрим случай, когда рабочая программа из мультипрограммного набора образовала последовательность вызова ввода-вывода и обратилась к исполнителю. В блоке 1 происходит блокировка прерываний, завершающих ввод-вывод. Это необходимо для того, чтобы исполнитель мог быстро произвести необходимую модификацию списков и очередей, которые отражают состояние рассматриваемой рабочей программы. Кроме того, определенное время пойдет на выполнение запроса ввода-вывода, после чего исполнитель будет вновь модифицировать соответствующие списки и очереди. Естественно, что в это время никаких прерываний, завершающих ввод-вывод, быть не должно. Вообще говоря, когда начинается модификация любого списка или очереди исполнителя, ни о каких прерываниях ввода-вывода не может быть и

речи до тех пор, пока указанная модификация не прекратится.

Блок 2 соответствует занесению в память содержимого регистров, индексов и т. д., которые использовались текущей обрабатываемой программой в тот

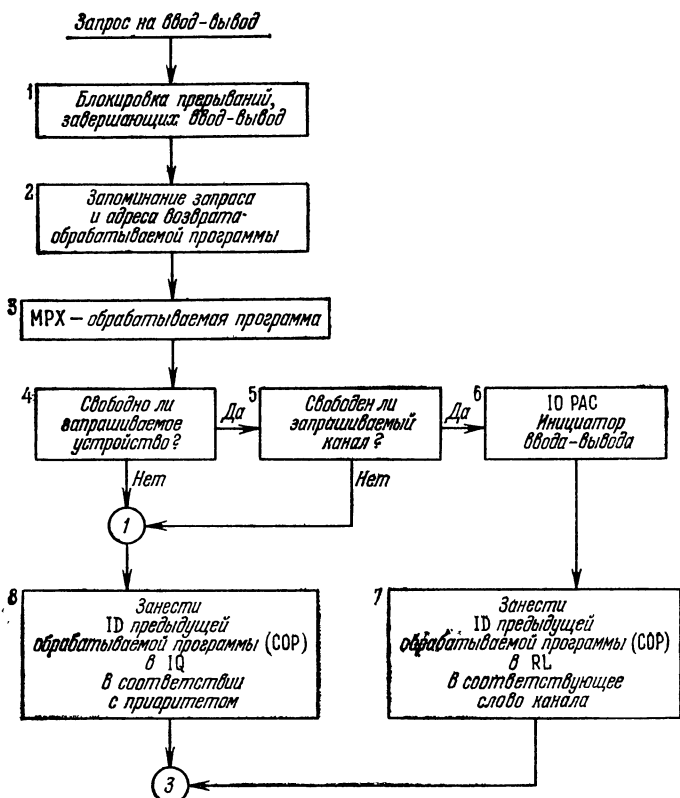


Рис. 3. Мультипрограммный Исполнитель. Блок-схема 1.

момент, когда она обратилась ко вводу-выводу. Это занесение осуществляет исполнитель. Он запоминает также и адрес возврата, по которому будет осуществляться переход к этой программе, в случае, если она будет вновь выбрана для текущей обработки. После этого мультипрограммный исполнитель сам становится текущей обрабатываемой программой (блок 3). Это

позволяет по каким-либо причинам осуществлять прерывания исполнителя, немедленно их обрабатывать и вновь передавать управление на то место исполнителя, на котором произошло прерывание. (Заметим, что такие прерывания обычно все же не связаны с управлением списками и очередями, которое осуществляет исполнитель.)

Далее (см. блок 4) анализируется, свободно или занято запрашиваемое устройство. Устройствами, занятыми в тот момент, когда канал свободен, могут быть магнитофоны с перематываемыми магнитными лентами, устройства памяти с произвольным доступом, выполняющие операции просмотра. Затем анализируется, свободен или занят запрашиваемый канал (блок 5). Если занятым окажется канал или устройство (или и канал и устройство), делается переход на связку 1. Так как ввод-вывод инициировать в этом случае нельзя, ID рабочей программы, которая запрашивала этот ввод-вывод, заносится в очередь на требуемый канал в соответствии с ее приоритетом (блок 8).

Возвратившись к анализу, который проводится в блоках 4 и 5, предположим, что свободны и устройство и канал. В этом случае (блок 6) делается переход на блок ввода-вывода, где интерпретируется последовательность вызовов, соответствующая данной программе, и после этого иницируется действие по вводу-выводу. Исполнитель заносит ID запрашивающей программы в список ссылок (в соответствующее слово состояния канала), показывая тем самым, что используемый рассматриваемой программой канал будет занят до окончания ввода-вывода. После этого делается переход на связку 3.

До связки 3 либо начался ввод-вывод для рабочей программы, либо она была поставлена в очередь, либо был осуществлен переход из состояния 3 в состояние 2 или 1. Дальнейшая задача МРХ заключается в выборе следующей текущей обрабатываемой программы из тех программ, которые находятся в состоянии 3. Такой выбор осуществляется, начиная с третьей связки.

В блоке 10 (рис. 4) отменяется запрет на прерывание ввода-вывода. В блоке 11 проводится проверка на наличие программ в очереди готовности. Если все программы поставлены в очередь IQ или ожидают



окончания ввода-вывода, то проверка будет циклически повторяться до тех пор, пока не произойдет прерывание, завершающее ввод-вывод. После прерывания одна из программ становится доступной для обработки. Затем эта программа проверяется в блоке 11, и завершающие ввод-вывод прерывания вновь запрещаются

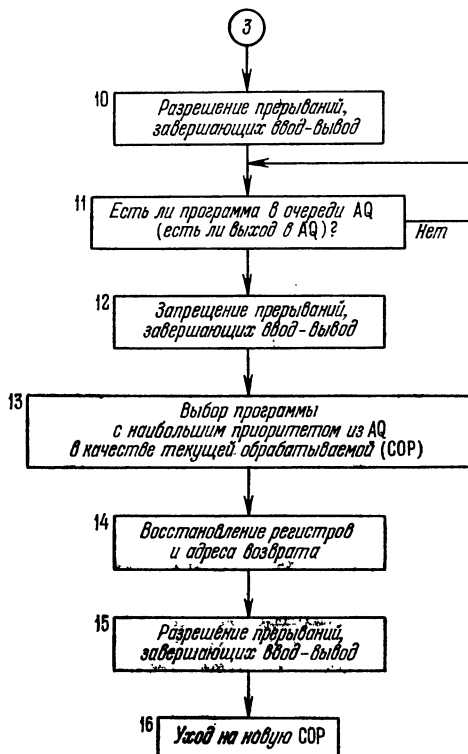


Рис. 4. Мультипрограммный Исполнитель. Блок-схема 2.

(блок 12). В обычном случае в очереди готовности стоит несколько программ, и после первой же проверки прерывания ввода-вывода запрещаются. После этого из очереди выбирается первая программа, т. е. программа, обладающая наивысшим приоритетом, и ставится на обработку (блок 13). Теперь эта программа становится текущей обрабатываемой. И, наконец, (блок 14) исполнитель извлекает из памяти значение индекс-ре-

гистров, которые запоминались при переходе ко вводу-выводу. Адрес ухода устанавливается в соответствии с адресом возврата, который запоминался для этой программы (блок 16). После этого снова разрешаются прерывания ввода-вывода, и управление передается на новую текущую обрабатываемую программу.

### **Завершение ввода-вывода**

Предположим теперь, что рабочая программа выполняется и в этот момент происходит прерывание, завершающее ввод-вывод через некоторый канал, сигнализируя о том, что действия по вводу-выводу, относящиеся к данному каналу, окончены.

После обработки этого прерывания управление передается на вход исполнителя, связанный с обработкой прерываний ввода-вывода, как показано на рис. 5. Начиная с блока 17, повторяется последовательность действий, которой придерживается исполнитель при вызове ввода-вывода (при запросе на ввод-вывод). Запрещаются прерывания, завершающие ввод-вывод, и запоминаются (упрятываются) индекс-регистры для текущей обрабатываемой программы, а также адрес возврата, на который исполнитель передает управление, когда программа будет выбрана для обработки. MPX становится текущей обрабатываемой программой и (в блоке 20) помещает ID, стоящий в слове состояния канала RL, связанном с освободившимся каналом (на котором произошло прерывание, завершающее ввод-вывод), в очередь готовности, сообразно с приоритетом программы. Чтобы указать на то, что канал свободен, указанное слово состояния канала RL обнуляется (блок 21).

Так как канал освободился, проводится проверка на наличие программ из очереди IQ, запрашивающих этот канал (блок 22). Если таковых не имеется, осуществляется переход на связку 3, и, как ранее было описано, производится выбор новой программы для текущей обработки.

Если же в очереди IQ есть программа, которая запрашивает данный канал, проводится проверка с целью определения, занято или свободно запрашиваемое устройство (блок 23). В том случае, если устройство не

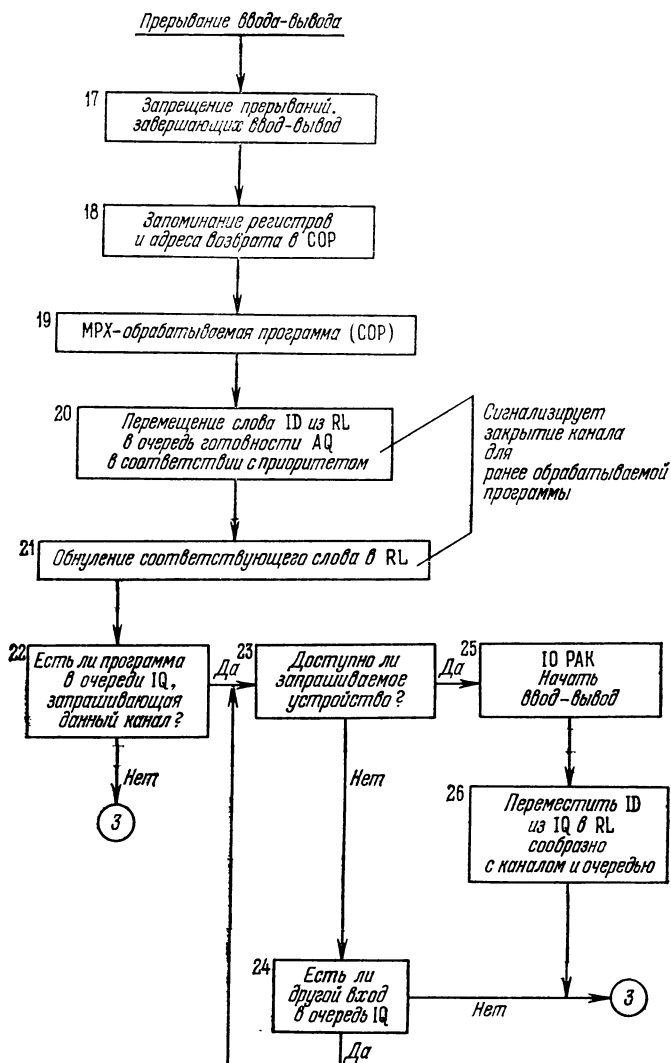


Рис. 5. Мультипрограммный Исполнитель. Блок-схема 3.

занято, очередь IQ просматривается вновь с тем, чтобы найти новую программу, которая запрашивала бы освободившийся канал. Если такая программа найдется, вновь будет проводиться проверка, позволяющая определить, свободно или занято запрашиваемое устройство. Этот циклический просмотр, сопровождаемый проверкой доступности устройства, будет проводиться до тех пор, пока не найдется программа, для которой можно инициировать ввод-вывод, и когда такая программа будет найдена (ею будет программа, для которой и канал и устройство свободны), управление передается на блок ввода-вывода (IO PAK — блок 25); ID этой программы будет выброшен из IQ и занесен в соответствующее слово RL (см. блок 26).

В любом из рассмотренных случаев, и в случае, когда иницируются действия по вводу-выводу, и в случае, когда этого не происходит, так как не найдено подходящей программы, для которой это можно сделать в данный момент, управление передается на связку 3, с тем чтобы выбрать новую программу для текущей обработки.

Именно эти два события: запрос на ввод-вывод и завершение ввода-вывода являются началом и концом полного цикла переходов из одного состояния в другое (т. е. трех состояний), которые может принимать программа, входящая в мультипрограммный набор.

Заметим, что, если программа запрашивает ввод-вывод, исполнитель старается выполнить этот запрос немедленно (т. е. немедленно инициировать ввод-вывод). Если сделать это не удается, идентификатор программы помещается в инициационную очередь и ввод-вывод иницируется сразу же, как только запрашиваемые устройство и канал освобождаются. Однако следует помнить, что все перечисленные действия выполняются в соответствии с системой приоритетов, и для того, чтобы ввод-вывод для данной программы иницировался, в первую очередь она должна обладать наивысшим приоритетом среди программ в IQ, которые запрашивают тот же самый канал.

Каждая из программ, обладающих менее высоким приоритетом, либо ожидает завершения ввода-вывода, либо находится в очереди IQ, ожидая инициации ввода-вывода.

## Завершение выполнения программы

Рабочая программа обращается к исполнителю в третий раз, когда выполнена ее задача.

При этом обращении состояние ее не меняется, причем в списки и очереди никаких изменений не вносится и никаких манипуляций с этими таблицами и очередями

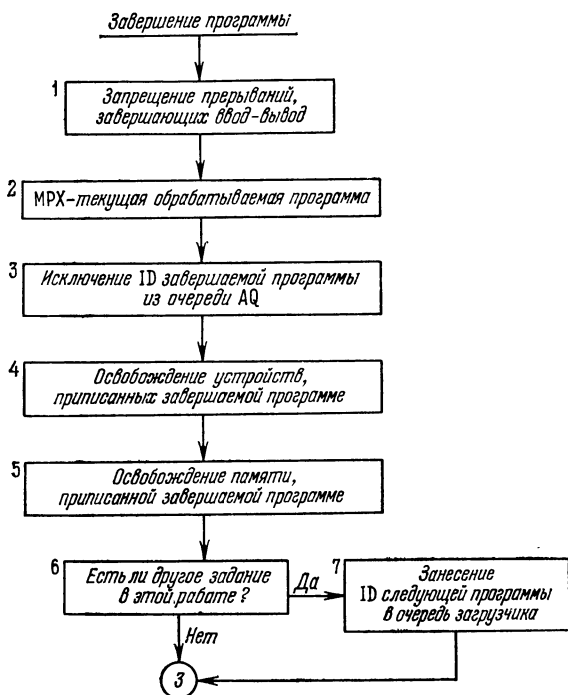


Рис. 6. Мультипрограммный Исполнитель. Блок-схема 4.

не производится. Исключением является лишь очередь AQ, в которую вносится лишь небольшое изменение. Блок-схема 4 (рис. 6) иллюстрирует ту последовательность действий, которой придерживается исполнитель при завершении выполнения программы. Как и в предыдущем случае, прерывания, завершающие ввод-вывод,

запрещаются, и МРХ становится текущей обрабатываемой программой. Далее ID завершаемой программы выбрасывается из очереди готовности (блок 3) (это фактически указывает на то, что выполнение программы заканчивается), и таким образом исключается возможность выбора этой программы в качестве текущей обрабатываемой. Так как AQ является лишь очередью, в которой стояла программа, устранение ID из нее влечет за собой еще и освобождение всей памяти, которая использовалась для хранения данной программы. Остальная часть блок-схемы иллюстрирует те действия, которые выполняет исполнитель при собственно завершении программы. Короче говоря, те устройства, которые были приписаны данной программе, освобождаются (возвращаются в исходное состояние, сбрасываются) и приписываются программам, входящим в систему (блок 4). Память, отведенная под завершennую программу, становится доступной для присвоения другим программам (блок 5). После этого управление передается на связку 3, и исполнитель выбирает новую программу для текущей обработки.

В отношении ввода-вывода исполнитель никаких действий не предпринимает, так как все возможные вводы-выводы уже инициированы до того момента, когда завершаемая программа была выбрана для текущей обработки. Следовательно, поскольку эта программа не затрагивает ввод-вывод и так как не происходит никаких прерываний ввода-вывода, состояние работ по вводу-выводу для всей системы остается неизменным.

### **Общие замечания**

Исполнитель, который мы рассмотрели в данной главе, построен в предположении, что все необходимые для программ действия по вводу-выводу осуществляются без буферизации (по отношению к этим программам). Именно это предположение позволяет нам рассматривать три взаимоисключающие состояния рабочей программы, которые исчерпывающе описывают все возможные варианты поведения такой программы в вычислительной системе и определяют логику работы исполнителя.

При построении мультипрограммного исполнителя, в случае небуферируемого ввода-вывода, мы использовали программные идентификаторы ID с тем, чтобы снабдить исполнителя необходимой информацией о программах. ID рассматривался нами как некий представитель программы в исполнителе (в тексте вместо термина «представитель программы» достаточно часто употреблялось слово «программа»). В частности, исполнитель манипулировал идентификаторами ID, заносил их в различные таблицы и очереди, которые использовались для контроля за составлениями программ. Таким образом, расположение ID в таблицах соответствовало некоторому состоянию, в котором находилась программа, описанная этим идентификатором. Описание программ с помощью идентификаторов (ID) представляет собой некую элементарную, но вряд ли функциональную, форму описания состояний рабочих программ в вычислительной системе. В следующей главе дается другая форма описания, отличающаяся от предыдущей большим совершенством и полнотой. ID в ней заменяются на запросы.

Это даст нам хорошую систему обозначений при сортировке и обеспечит логическую полноту, которая необходима при рассмотрении ряда параллельных действий в мультипрограммировании.

## ГЛАВА 5

### ЗАПРОС ЦЕНТРАЛЬНОГО ПРОЦЕССОРА

#### Введение

Результаты выполнения программы имеют большую важность для программиста. Для того, чтобы добиться этих результатов в каждом отдельном случае, программа должна иметь такую логическую организацию, которая позволяла бы манипулировать с данными, имеющими некоторую специфическую структуру. Таким образом, задача, решение которой осуществляется с помощью рабочей программы, определяет и логическую организацию программы и структуру данных, с которыми она работает. Поэтому каждый программист должен четко понимать, какая программа может быть выполнена системой.

Вообще, можно было бы сказать, что ОС работает с рабочими программами, как с данными, т. е. основная задача ОС как раз и заключается в том, чтобы манипулировать рабочими программами, и с этой точки зрения ОС совершенно не зависит от результатов выполнения каждой рабочей программы в отдельности.

Иными словами, логическая организация операционной системы, в отличие от логики рабочих программ, ориентированных на решение некоторых частных задач, основана на принципах, которые позволяют обеспечить любую программу, независимо от задачи, решаемой с ее помощью, всеми средствами, необходимыми для ее выполнения. Наконец, для операционной системы совершенно не важно, какую задачу призвана



решить та или иная программа, и в этом смысле все рабочие программы для нее одинаковы.

Для того, чтобы выделить программу из общей массы, используется система приоритетов, с помощью которой операционная система может отдать предпочтение той или иной рабочей программе. Приоритет программы заносится в некоторое отведенное место памяти, которое просматривается при выборе текущей обрабатываемой программы, и на основании принятых критериев система выбирает для обработки программу, обладающую высшим приоритетом среди всего набора программ, резидентных в системе.

## Поля запросов

Из сказанного выше видно, что работа ОС фактически не зависит от назначения каждой рабочей программы в отдельности. Для нее имеют значение лишь некоторые специфические данные описательного характера, связанные с каждой из программ. Например, для того, чтобы обработать рабочую программу, операционная система должна проанализировать некоторую информацию об этой программе с тем, чтобы выполнить над ней все необходимые операции (ввод-вывод, обработку на ЦП и т. д.). Естественно, что для каждой рабочей программы такая информация должна быть где-то сформирована.

Адрес ячейки, начиная с которой загружается программа, объем занимаемой памяти, идентификаторы устройств, которые приписаны данной программе, наряду с информацией о распределении файлов по этим устройствам, — все это относится к описательной информации, которая должна формироваться для рабочих программ. Кроме того, операционная система должна располагать определенными данными о программе, выполнение которой необходимо возобновить после прерывания.

К этим данным относится команда, с которой должна возобновиться обработка программы (адрес возврата), а также значения индекс-регистров в момент прерывания. Для того, чтобы хранить такие данные, необходимо отвести достаточное количество памяти и предусмотр-

реть использование нужного количества индекс-регистров.

Ясно, что такая информация обладает однозначностью и позволяет в каждом отдельном случае проанализировать состояние рабочей программы. Фактически для каждой рабочей программы как бы составляется досье, в котором отражаются все те события, которые с ней происходят, а также содержится необходимый справочный материал. Естественно, что при таком подходе все программы будут обладать досье, отражающими в каждый момент времени состояние их обработки. В дальнейшем такие досье мы будем называть *полями запросов* (для данного состояния обработки данной программы).

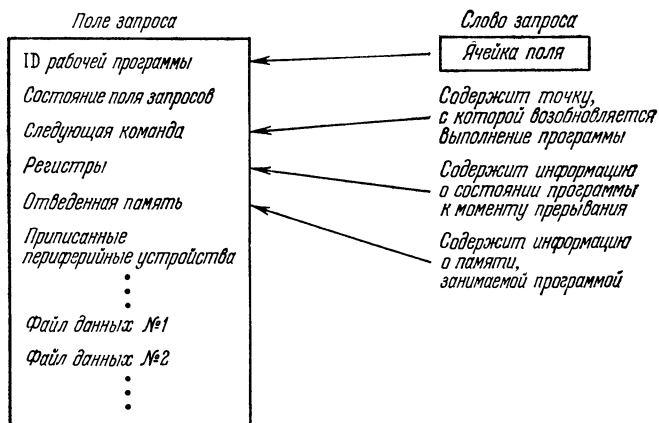


Рис. 7. Структура поля запросов.

На рис. 7 показан пример структуры поля запросов, с указанием некоторых данных, которые могут в нем содержаться. Предполагается, что эти данные заносятся в ячейки памяти, расположенные рядом друг с другом. В общем случае такое ограничение на поля запросов не накладывается, однако данные необходимо должны быть логически совместимыми.

Это достигается за счет задания адреса ячейки, начиная с которой располагается каждая часть территориально разобщенного поля запроса. В результате

этого каждая предыдущая его часть получает связь с последующей.

Таким образом, задавая адрес начала (или имя) той или иной части поля запроса, мы можем обеспечить логическую совместимость и сделать доступной (извлечь) всю информацию, помещенную в нем.

Как уже отмечалось ранее, задача операционной системы состоит в том, чтобы управлять обработкой рабочих программ, причем для того, чтобы осуществлять это управление, система должна использовать некоторые специфические данные описательного характера, которые относятся к каждой из рабочих программ. Ими-то (данными) как раз и являются данные, заносимые в поля запросов. Именно ими операционная система пользуется при анализе состояний рабочих программ. Таким образом, поле запросов снабжает операционную систему всей необходимой информацией о любой рабочей программе. Фактически операционная система манипулирует полями запросов точно так же, как идентификаторами программ.

Чтобы несколько упростить поиск полей запроса и манипуляцию ими, мы будем давать каждому полю свое имя.

### Слово запроса

Как уже отмечалось выше, требование логической совместимости обуславливает необходимость указывать адрес (или идентификатор) первой или начальной ячейки поля запроса. Поскольку поля запросов размещаются в оперативной памяти, первая ячейка каждого из них должна иметь свое собственное имя (идентификатор), отличающееся от всех остальных. Следовательно, первую ячейку поля запроса мы будем считать именем этого поля или его идентификатором (ID). В дальнейшем мы будем называть ее *словом запроса*.

Слово запроса представляет собой уже некую единицу информации, с которой удобно работать операционной системе. Это слово не имеет постоянного места в памяти и может размещаться в различных ее ячейках, в зависимости от того, куда его поместила операционная система. В этом слове содержится начальный адрес поля запроса, т. е. тот адрес, начиная с которого

последовательно размещается собственно само поле запроса. Следовательно, слово запроса является как бы представителем соответствующего поля запроса в операционной системе. А так как поле запроса имеет собственное имя и является неким досье рабочей программы, то слово поля запроса представляет в операционной системе рабочую программу.

### Запрос ЦП

Известно, что программа представляет собой последовательность команд. Для выполнения программы требуется использование ЦП, т. е. он должен на некоторое время как бы приписываться данной программе и обрабатывать ее. С тем, чтобы отличить данную работу от других, параллельно с ней выполняемых, необходимо связать ее с соответствующим полем и словом запроса. Работу процессора можно представить себе как некий процесс «просмотра» команд программы, причем на то, чтобы просмотреть каждую команду, требуется некоторое количество времени. Выполнение программы или же ее просмотр, который осуществляет ЦП, непосредственно связаны с полем и словом запроса, являющимися представителями программы в системе. Поэтому мы можем сказать, что запрос «просматривает» или «выполняет» эту программу. А так как этот запрос непосредственно связан с выполнением команд и приписыванием ЦП для выполнения этих команд, мы будем называть его *запросом* ЦП. Таким образом, запрос ЦП служит для отражения того, что для заданной программы необходимо использование ЦП, а также для того, чтобы указать работу, с которой этот запрос связан. Использование запроса ЦП подробно будет описываться в главе 7. По аналогии с запросом ЦП строится и запрос ввода-вывода, который отражает необходимость использования периферийных устройств для данной программы. Этот второй тип запроса рассматривается в главе 8.

### Поле ЦП

Введем понятие *текущий обрабатываемый запрос*. Так мы будем называть тот запрос, в результате анализа которого ЦП обрабатывает связанную с ним программу.

Естественно, что запрос ЦП должен иметь некую информационную часть, которая может анализироваться операционной системой, а также и модифицироваться ею с тем, чтобы обеспечить использование ЦП тогда, когда этот запрос становится текущим обрабатываемым. В связи с этим информационная часть запроса ЦП, назовем ее *полем ЦП*, размещается в оперативной памяти и состоит из двух подполей, имеющих длину, достаточную для занесения адресов ячеек оперативной памяти. Структура поля ЦП показана на рис. 8.



Рис. 8. Структура поля ЦП.

В первом подполе — COT (Current Operating Transaction field) содержится слово запроса для текущего обрабатываемого запроса. Имя программы, которая в данный момент использует ЦП, заносится именно в это подполе. Во время работы вычислительной системы в подполе COT обязательно находится какое-нибудь слово запроса. Если обрабатывается рабочая программа, то в это поле записывается ее слово запроса, а если обрабатывается сама операционная система (или некоторый ее модуль, имеющий собственное слово запроса), то в поле COT попадает уже слово запроса самой операционной системы (или ее модуля). Это подполе COT мы будем называть *подполем текущего обрабатываемого запроса*.

Второе подполе предназначается для слова запроса, с которым работает программа, представленная в ОС текущим обрабатываемым запросом. Это подполе может быть и пустым. Будем называть его *подполем готовности запроса* AT (Available Transaction field).

Как мы уже говорили, поле ЦП служит для присвоения ЦП той или иной работе. Это достигается занесением слова запроса текущего обрабатываемого запроса в первое подполе COT поля ЦП. Определенные запросы, например, запросы операционной системы, будут выполнять инструкции, назначение которых состоит в

том, чтобы манипулировать другими запросами (обычно представляющими рабочие программы). Подполе АТ поля ЦП используется как системный регистр, помогающий в осуществлении такой манипуляции.

Для того, чтобы сделать более понятным определение поля ЦП, которое является одним из важнейших элементов при разработке наиболее общего подхода к операционной системе, покажем его на конкретном примере. На рисунке 9 изображено поле ЦП, и для некоторого частного случая показаны его связи с полями запросов в оперативной памяти. Это делается через входы в подполя СОТ и АТ. В качестве имени текущего обрабатываемого запроса выбрано число 3000 (рис. 9, а).

Это имя заносится в подполе СОТ поля ЦП. Т. е. в подполе СОТ указывается номер ячейки, с которой начинается группа ячеек для отдельно взятого поля запроса (в нашем случае указанная группа ячеек, начинающаяся с ячейки 3000, и обозначена как поле запроса 1). Этому запросу и, следовательно, связанной с ним рабочей программе дано символическое имя *W1*. (На рисунках символические имена рабочих программ подчеркнуты.)

На рис. 9, б показан случай, когда текущим обрабатываемым является запрос с именем 3040. Это начальный адрес поля запроса 3000 с символическим именем *Z*. Кроме того, отражена ситуация, когда запрос *W1* (с начальным адресом 3000), ранее рассматривавшийся в качестве текущего обрабатываемого, перешел в состояние готовности и, в связи с этим, попал во второе подполе АТ поля ЦП. Таким образом, на рис. 9, б показано, что запрос *Z* стал текущим обрабатываемым в то время, как запрос *W1* перешел в состояние готовности и находится в нем. Это означает, что ЦП

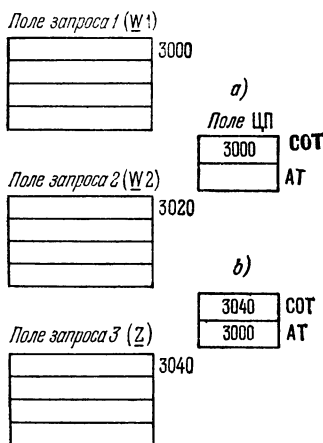


Рис. 9. Поля запросов в оперативной памяти.

приписывается программе, представленной запросом  $Z$ , и указывает на то, что запрос, который будет активизировать  $Z$ , в данный момент является запросом  $W1$ , находящимся в состоянии готовности.

## Функция ОС

До сих пор мы связывали работу программы с ее запросом. Запрос состоит, как мы уже говорили, из двух частей. Одна является информационной и содержит всю необходимую информацию для управления выполнением программы — это поле запроса. Другая часть является как бы именем программы или поля запроса и удобна для манипуляции — это слово запроса.

Кроме того, мы полагали, что программа выполняется на основании просмотра запроса операционной системой.

С другой стороны, запрос является как бы отражением состояния обработки программы в любой момент времени. Поэтому можно сказать, что запрос «выполняет» программу или просматривает ее. В связи с этим можно полагать, что определенные логические функции выполняются запросом за счет того, что этот запрос сам подвергается действию некоторой логической функции.

Такую функцию мы сейчас и рассмотрим. Назовем ее *входной граничной функцией операционной системы* (сокращенно *функцией ОС*). Заметим, что она лежит в основе нашего подхода к построению операционной системы. Ее назначение заключается в том, чтобы различать, какими программами используются средства вычислительной системы: прикладными или программами операционной системы. Т. е. обработка рабочей программы выполняется вне входной границы ОС, а обработка самой ОС проходит во внутренней области этой границы. Для того, чтобы перейти к обработке ОС от обработки рабочей программы, потребуется пересечь эту границу ОС. Этот переход через границу осуществляет запрос рабочей программы, который в данный момент является текущим обрабатываемым (СОТ). Логическая операция, соответствующая функции ОС, заключается в том, чтобы перевести текущий обрабатываемый запрос (для рабочей программы) в состояние готовности

(АТ), а в качестве текущего обрабатываемого выбрать требуемый запрос ОС.

Рис. 10, а в схематической форме иллюстрирует прохождение через входную границу ОС. Входящая и выходящая стрелки показывают направление, в котором перемещается текущий обрабатываемый запрос по мере того, как он «проходит» через выполняемые команды, причем не имеет значения, какой запрос является текущим обрабатываемым. На рис. 10, а показано также поле ЦП до и после прохождения запроса через функцию

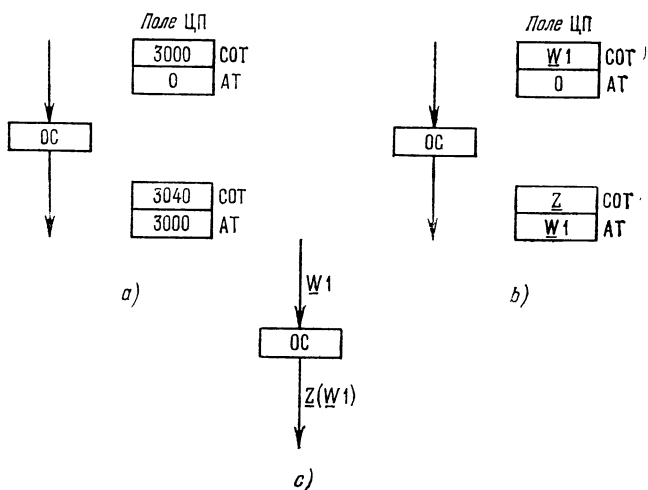


Рис. 10. Переход через входную границу операционной системы.

ОС, т. е. инструкции, которые обеспечивают выполнение этой функции. До прохождения через функцию ОС в поле ЦП текущим обрабатываемым запросом был запрос рабочей программы с именем 3000. Он попал в поле ЦП в результате предыдущего выполнения функции ОС и прохождения через входную границу ОС.

После выполнения функции ОС запрос рабочей программы перешел в состояние готовности, а его место в поле ЦП занял запрос с именем 3040, став текущим обрабатываемым. Как мы предполагали ранее, поле запроса, начинающееся в оперативной памяти с ячейки 3040, связано с обработкой программы ОС. Таким образом, входная граница ОС пересечена, и запрос рабочей



программы, которая инициировала это перемещение, перешел в состояние готовности, а запрос ОС с именем 3040 стал текущим обрабатываемым (COT).

Именно этот последний запрос пройдет через программу ОС, манипулируя запросом АТ и другими запросами так, как это задано логикой программы для выполнения требуемых функций ОС.

Рис. 10, *b* эквивалентен 10, *a* с той лишь разницей, что имя запроса рабочей программы мы заменили символом *W1*, а имя запроса операционной системы обозначили *Z*. До входа в функцию ОС *W1* был текущим обрабатываемым запросом и представлял рабочую программу до перехода входной границы ОС.

После того, как граница была пройдена в результате выполнения функции ОС, *W1* был переведен в состояние готовности и, следовательно, занесен в поле АТ поля ЦП. В то же время запрос *Z* стал текущим обрабатываемым и попал в поле COT поля ЦП.

Рис. 10, *c* отражает ту же ситуацию, что и два предыдущих рисунка, однако в нем была использована система обозначений, более удобная при построении блок-схем для запросов, которые будут показаны ниже.

На этом рисунке вновь текущим обрабатываемым запросом является запрос рабочей программы *W1*. Для того, чтобы показать это, мы поставили рядом со стрелкой, входящей в функцию ОС, обозначение *W1*.

Текущим обрабатываемым запросом становится *Z*, что указывается путем помещения обозначения *Z* рядом с выходящей стрелкой. Обозначение *W1* ставится в круглые скобки и помещается после *Z*. Это означает, что запрос *W1* переходит в состояние готовности.

Запись  $A(B)$ , где *A* является запросом в COT, а *B* — запросом в АТ, будем называть *термом запросов*.

### Модуль возврата

Если функция ОС представляет собой входную границу в операционную систему, то модуль возврата (RETURN) служит в качестве выходной границы. Так как та программа, которая обрабатывалась до пересечения выходной границы, была программой операционной системы, модуль возврата удаляет запрос операционной системы *Z* из подполя COT поля ЦП, занося

в это поле слово следующего запроса, выбранного в качестве текущего обрабатываемого. На место  $Z$  попадает содержимое подполя АТ поля ЦП, т. е. следующим текущим обрабатываемым будет запрос, чье имя стояло в подполе АТ поля ЦП. Эти операции иллюстрируются на схематической диаграмме (рис. 11, а). До вхождения

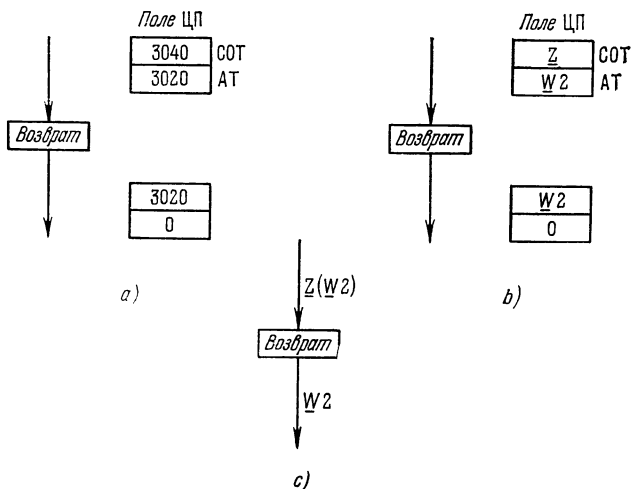


Рис. 11. Прохождение через выходную границу операционной системы.

в блок возврата в подполе COT поля ЦП стояло слово запроса 3040.

В процессе выполнения программы операционной системы сама ОС выбрала в качестве текущего обрабатываемого запрос рабочей программы 3020 на оборудование. Вследствие этого он появился в подполе АТ поля ЦП. После выполнения модуля возврата запрос операционной системы (3040) в подполе COT поля ЦП заменяется запросом 3020, взятым из подполя АТ. После этого подполе АТ обнуляется.

На рисунке 11, b дана символическая интерпретация ситуации, рассмотренной выше. До входа в модуль возврата подполя COT и АТ содержали запрос операционной системы  $Z$  и, выбранный в качестве следующего обрабатываемого, запрос  $W2$ , соответственно. После прохождения выходной границы запрос ОС был

заменен в подполе COT запросом  $W_2$ , после чего подполе AT было обнулено.

На рисунке 11, с показано то же самое прохождение выходной границы (или модуля возврата), но с использованием системы обозначений, которой мы будем придерживаться в дальнейшем при построении карт или схем запросов. Символическое обозначение запроса  $Z$  операционной системы ставится рядом (справа) со стрелкой, которая обозначает вход в модуль возврата. Запрос  $W_2$  поставлен в скобках в соответствии с тем, что  $W_2$  является запросом в состоянии готовности. Рядом с выходной стрелкой стоит только  $W_2$ , и это означает, что  $W_2$  заменил собой  $Z$  в подполе COT поля ЦП.

### Схема запросов

Принцип построения схемы запросов, в основном, такой же, как и при изображении блок-схем программы, однако имеются и некоторые отличия. В частности, наряду с условными переходами и смысловыми блоками на схеме запросов указываются еще и текущие обрабатываемые запросы, а также запросы в состоянии готовности. Кроме того, как мы впоследствии увидим на схеме запросов МРХ, наибольшую важность представляют логические операции, связанные с манипуляцией запросами. Таким образом, сам по себе метод определения того, свободно ли требуется устройство для инициализации соответствующих операций ввода-вывода, будет интересовать нас в значительно меньшей степени, нежели то место в блок-схеме, где такой анализ надо провести, и как это отразится на запросах, указанных в терме ЦП.

### Схема запросов для монитора (МРХ)

На рис. 12 показана схема запросов МРХ для случая небуферируемого запроса на ввод-вывод (который был описан в главе 4). Предполагается, что рабочая программа представлена текущим обрабатываемым запросом или анализируется *мультипрограммным исполнителем (монитором)* (МРХ) после очередного прерывания ввода-вывода.

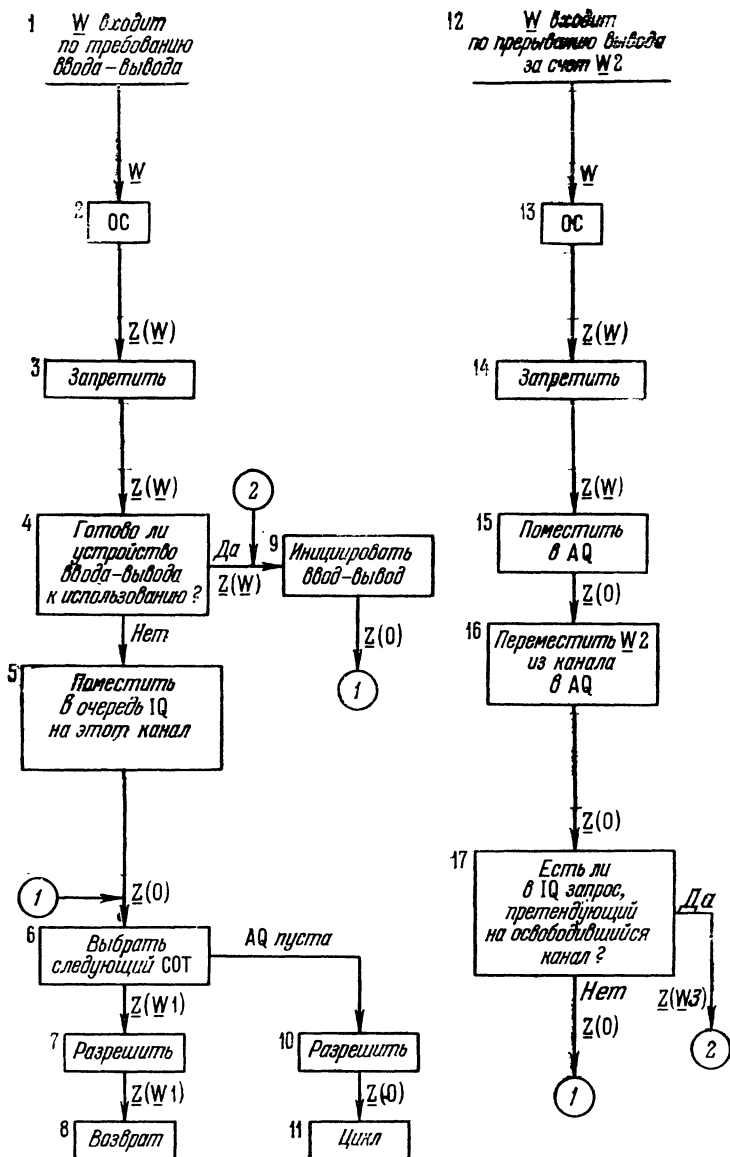


Рис. 12. Схема запросов для МРХ.

Рабочая программа обращается в МРХ по запросу на ввод-вывод в блоке 1. Имя рабочей программы, т. е. ее слово запроса, мы обозначим буквой  $W$ . Так как при обращении к операционной системе  $W$  был текущим обрабатываемым (СОТ), то мы и поставим его рядом со стрелкой, входящей в ОС. Таким образом, текущим обрабатываемым запросом, который пересекает границу ОС, становится  $W$  (см. блок 2). После выполнения модуля ОС, как уже говорилось ранее, текущим обрабатываемым (СОТ) будет уже запрос операционной системы  $Z$ , а  $W$  перейдет в состояние готовности.

Запрос  $Z$  теперь уже проходит через МРХ, и первой его функцией будет запрет дальнейших прерываний ввода-вывода. При запрете прерываний поле ЦП останется неизменным, и поэтому на выходе из блока 3 терм ЦП также не изменяется:  $(Z(W))$ , т. е.  $Z$  продолжает оставаться текущим обрабатываемым, а  $W$  находится в состоянии готовности.

В блоке 4 определяется, может ли в данный момент выполняться требуемый для  $W$  ввод-вывод. Конкретная реализация такого анализа зависит от характера запроса на ввод-вывод, а также от аппаратной конфигурации системы, в которой функционирует МРХ. Так, например, готовность к выполнению ввода-вывода (I/O READY \*) в одной системе определяется тем, что свободны соответствующие канал и устройство, в то время как в другой системе, использующей коммутатор каналов (мультиплексор), критерием готовности может служить только доступность устройства.

Независимо от того, в какой системе это имеет место, важно, что вопрос: можно ли осуществлять ввод-вывод вообще? — задается и, что самое главное, задается в отношении запроса, находящегося в подполе АТ поля ЦП.

Поскольку готовность к выполнению ввода-вывода должна анализироваться (или, иначе говоря, должна выполняться функция I/O READY), необходимые для этого данные помещаются в поле запроса, которое соответствует запросу в состоянии готовности (т. е. слову запроса, находящемуся в подполе АТ поля ЦП).

---

\*) Ввод и вывод по-английски — Input и Output, сокращенно — I/O.

Иными словами, можно сказать, что  $Z$  выполняет подпрограмму I/O READY, причем  $W$  является элементом в последовательности ее вызова (т. е. неким формальным параметром). При такой интерпретации  $W$  аналогично косвенному адресу для конкретной последовательности вызова.

Независимо от того, какая из двух возможностей будет реализована с помощью функции I/O READY, терм ЦП (т. е.  $Z(W)$ ) остается прежним.

В случае, если зафиксирована готовность к вводу-выводу для  $W$ , дальнейшие действия выполняются в блоке 9. Терм ЦП показывает состояние поля ЦП на входе в блок 9. Этот терм идентичен терму на входе в блок 4, и функция INITIATE I/O (инициировать ввод-вывод), так же как и при выполнении функции I/O READY, будет работать с содержимым подполя AT поля ЦП, рассматривая его как косвенный адрес для вызова.

Таким образом, модуль, выполняемый в блоке 9, обращается к тем данным поля запроса  $W$ , которые имеют отношение к выполнению требуемого ввода-вывода. И вновь, как и в случае функции I/O READY, фактическая реализация модуля INITIATE I/O зависит от конкретного оборудования. Особенно важным при выполнении модуля инициирования ввода-вывода является то, что терм ЦП изменяется. В частности, как мы уже указывали в главе 4, инициирование небуферизуемого ввода-вывода для каждой отдельно взятой программы должно сопровождаться действиями по занесению ID программы в соответствующее слово списка ссылок RL, в котором этот ID будет находиться до завершения ввода-вывода. Теперь, когда мы ввели понятие запроса, поля запроса и слова запроса, становится ясно, что функции ID программы выполняет слово запроса. Отсюда вытекает, что блоком 9 выполняется еще и некая дополнительная работа, заключающаяся в перемещении слова запроса из подполя AT поля ЦП на соответствующее место в списке ссылок. Подполе AT в этом случае обнуляется, и иницируется ввод-вывод. После этого выполняются все операции, необходимые для модуля INITIATE I/O. Так как запрос  $W$ , находившийся в состоянии готовности, удаляется из подполя AT поля ЦП, терм ЦП на выходе из блока 9 претерпевает

изменения. Это изменение заключается в том, что в круглых скобках термина ЦП вместо  $W$  стоит нуль, что свидетельствует об отсутствии слова запроса  $W$  в подполе АТ поля ЦП. Текущим обрабатываемым запросом (СОТ) по-прежнему остается  $Z$ , т. е. операционная система. После выполнения всех функций блока 9 управление передается на связку 1.

Теперь вернемся к блоку 4 и предположим, что запрос (терм  $Z(W)$  на входе в блок 4) операции ввода-вывода обрабатывать нельзя, так как не выполняется критерий готовности. В этом случае терм ЦП не претерпевает никаких изменений, так как  $W$  так и остается в состоянии готовности к манипуляциям, которые может выполнять над ним текущий обрабатываемый запрос  $Z$ . В таком виде он и будет поступать на вход блока 5. Поскольку в данный момент ввод-вывод, требуемый для  $W$ , не может быть выполнен, и в силу того, что буферизация не осуществляется, обработка рабочей программы, представленной запросом  $W$ , откладывается до тех пор, пока не появится возможность инициировать этот ввод-вывод. Как было отмечено в главе 4, в которой мы рассматривали принцип построения мультипрограммного исполнителя, запрос, ожидающий освобождения устройства, помещается в очередь IQ на этот канал и удаляется из нее, когда канал становится доступным. Именно эту функцию и выполняет блок 5.

Терм ЦП изменяется в связи с тем, что запрос, находившийся в подполе АТ поля ЦП, удаляется из этого подполя и заносится сообразно с системой приоритетов в очередь IQ. Подполе АТ обнуляется, и терм становится таким, каким он показан на выходе из блока 5.

Выходная информация из блока 5, через связку 1, попадает в блок 6. На входе в блок 6 запрос  $Z$  должен быть текущим обрабатываемым, причем в подполе АТ поля ЦП должны содержаться только нули. Этим требованиям удовлетворяет также и терм ЦП на выходе блока 9.

Вхождение в блок 6 предъявляет к входным термам перечисленные выше требования. Каждое вхождение в этот блок из любой части ОС требует их удовлетворения.

Функцией блока 6 является выбор запроса (т. е. рабочей программы) из очереди готовности АҚ для

текущей обработки. Иными словами, в блоке 6 выбирается новый запрос для текущей обработки (COT). Входной терм  $Z(0)$  трансформируется в блоке 6 и появляется на его выходе в виде  $Z(WI)$ , где  $WI$  — новое слово запроса, стоящее в подполе АТ поля ЦП.

После этого терм с новым запросом для текущей обработки (COT) проходит через блок 7, в котором разрешаются прерывания, и попадает в блок 8 — модуль возврата (RETURN).

Модуль RETURN осуществляет прохождение через выходную границу ОС, и  $WI$  становится текущим обрабатываемым, обработка центральным процессором программы, связанной с этим словом запроса, возобновляется.

Теперь возвратимся к блоку 6, в котором осуществляется выбор запроса для текущей обработки, и предположим, что очередь AQ пуста. Это может произойти либо в том случае, если все запросы ожидают начала ввода-вывода и стоят в очереди IQ, либо все они ожидают окончания ввода-вывода, который уже инициирован для них. Однако в любом из этих случаев, связанных с выполнением ввода-вывода для запросов, хотя бы один из запросов в конечном счете обязательно возвратится в очередь AQ. Следовательно, МРХ должен содержать блок, разрешающий прерывания (блок 10), и блок, ждущий прихода прерывания, по выполнении которого в очередь AQ попадает слово запроса (блок 11). Заметим, что на входах блоков 10 и 11 термы идентичны, и подполе АТ соответствующего им поля ЦП обнулено. Это обусловлено тем, что выбор COT в данной ситуации невозможен, так как очередь AQ пуста.

Рассмотрим теперь последовательность действий, которые осуществляет МРХ, в случае, если происходит прерывание ввода-вывода. Предположим, что прерывание произошло по требованию запроса  $W2$ . Запрос, являющийся текущим обрабатываемым, для удобства дальнейшего рассмотрения назовем буквой  $W$ . Фактически в блоке 12 осуществляется вход в блок МРХ, обрабатывающий прерывание, причем  $W$  является текущим обрабатываемым. Как уже отмечалось выше, запросом, обуславливающим прерывание, является  $W2$ . При переходе входной границы ОС (блок 13) текущим обрабатываемым запросом становится



ся  $Z$ , а прерванный запрос  $W$  помещается в подполе АТ поля ЦП. В блоке 14 дальнейшие прерывания ввода-вывода запрещаются, и уже в блоке 15 запрос  $W$ , находящийся в состоянии готовности, так как он находится в состоянии 3, помещается в очередь АҚ. На выходе из блока 15 подполе АТ поля ЦП обнуляется. В блоке 16 ищется ячейка списка ссылок, которая соответствует данному прерыванию. Слово запроса  $W2$ , для которого закончился ввод-вывод, удаляется из списка ссылок и перемещается в очередь готовности, так как это слово представляет в системе рабочую программу, для которой ввод-вывод уже выполнен и которая может теперь перейти в состояние 3. Это не влияет, однако, на терм ЦП, который на выходе из блока 16 по-прежнему сохраняет вид  $Z(O)$ . В блоке 17 определяется наличие в очереди запросов IQ, претендующих на освободившийся канал. В случае отсутствия таких запросов действия по вводу-выводу инициированы не будут и, естественно, не будет выбран запрос, с которым мог бы работать СОТ, т. е. ОС.

В силу этого осуществляется переход на связку 1, причем терм ЦП полностью удовлетворяет требованиям, предъявляемым ко входным термам блока 6.

Если же в очереди IQ будет найден запрос, претендующий на освободившийся канал, слово этого запроса выбирается в качестве слова запроса в состоянии готовности и заносится в подполе АТ поля ЦП. Таким образом, терм ЦП на выходе блока 17 будет иметь вид  $Z(W3)$ , где  $W3$  — запрос, выбранный из соответствующей очереди. После выполнения функции блока 17 делается переход на связку 2, т. е. на блок 9.

Заметим, что выходной терм  $Z(W3)$  удовлетворяет всем требованиям, которые предъявляются ко входным термам блока 9. В этом блоке иницируется ввод-вывод, требуемый для  $W3$ . Слово запроса  $W3$  удаляется из подполя АТ и помещается в соответствующее слово списка ссылок, а подполе АТ поля ЦП обнуляется (см. рис. 12, блок 9). После этого осуществляется переход на связку 1 и, следовательно, на блок 6, где запрос ( $Z$ ) операционной системы выбирает следующий СОТ.

Мы завершим наше описание карты запросов для МРХ рассмотрением блока завершения программ.

Карта запросов для этого блока операционной системы показана на рис. 13. Пусть рабочая программа, выполнение которой заканчивается, имеет символическое имя  $W$ . При завершении рабочая программа обращается к МРХ и, действуя как СОТ, пересекает входную границу операционной системы. При пересечении границы (т. е. выполнении функции ОС) текущим обрабатываемым становится запрос операционной системы ( $Z$ ), а  $W$  попадает в подполе АТ поля ЦП. После этого операционная система, пользуясь словом запроса  $W$  как указателем для соответствующего поля запроса, освобождает память и устройства, закрепленные за данной программой, и обнуляет подполе АТ. Теперь МРХ может использовать освободившуюся память и устройства для закрепления за вновь поступившей в систему рабочей программой. Заметим, что на выходе из блока 3 текущим обрабатываемым остается запрос операционной системы  $Z$ , в то время как поле АТ пусто. Запрос находится в состоянии готовности. В связи с этим управление передается на связку 1 рис. 12, а впоследствии и на блок 6, где осуществляется выборка следующего СОТ из очереди готовности. Следует подчеркнуть, что и в этом случае терм ЦП ( $Z(0)$ ) удовлетворяет всем требованиям, которые предъявляются ко входным термам блока 6.

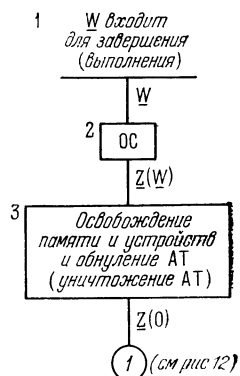


Рис. 13. Завершение программы.

### Общие замечания

Проводя анализ функционирования МРХ, работающего без буферизации, на языке запросов, мы показали возможности этого языка и его удобство при описании операционной системы. В дальнейшем мы также будем использовать язык запросов, рассмотренный выше с тем, чтобы сохранить единообразие в изложении принципов организации операционной системы. Понятия поля ЦП, его подполей СОТ и АТ, а также

символическая терминология запросов будут использоваться нами как основные инструменты в описании и анализе структур ОС, которые будут рассматриваться нами в следующих главах.

Действенность анализа запросов становится по существу очевидной при описании так называемых *модуля ввода и модуля вывода* операционной системы. С помощью этого метода мы можем описать независимую асинхронную работу различных функциональных модулей систем мультипрограммирования в единообразной и понятной читателю форме, которая при этом сохраняет всю динамику, содержащуюся в этих системах.

## ГЛАВА 6

### ОСНОВНЫЕ МОДУЛИ ОС

В главе 4 мы рассмотрели структуру монитора (МРХ), построенного в предположении, что операции ввода-вывода для рабочих программ осуществляются без буферизации. В главе 5 было дано определение запроса ЦП, и на основе метода запросов была проанализирована структура монитора при той же предпосылке небуферизируемого ввода-вывода. Иными словами, одна и та же структура монитора описывалась с помощью двух методов, последний из которых является более общим и будет использоваться для описания других модулей операционной системы. Таким образом, главы 5 и 4 были, в основном, посвящены рассмотрению лишь одного модуля операционной системы, и ни одного слова не было сказано об остальных ее составляющих.

Монитор независимо от того, к какой системе мультипрограммирования он относится (с буферизацией или без буферизации ввода-вывода), играет главенствующую роль в логической иерархии модулей операционной системы.

Это связано с тем, что именно МРХ распределяет ЦП и устройства ввода-вывода по запросам рабочих программ из мультипрограммного набора. В относительно простом случае МРХ для мультипрограммирования небуферизируемых запросов на ввод-вывод, распределения ЦП и устройств ввода-вывода, а также обработки прерываний, завершающих ввод-вывод, обуславливает переход нескольких запросов из мультипрограммного набора в три возможные состояния. Этот

переход из одного состояния в другое обуславливает, в свою очередь, управление очередями, которое осуществляет некоторый модуль МРХ, и в связи с этим ясно, что обработка самого МРХ, осуществляемая ЦП, не должна останавливаться из-за прерываний ввода-вывода. Именно по этой причине МРХ занимает центральное положение в операционной системе.

В настоящей главе мы рассмотрим остальные модули ОС, которые можно считать вспомогательными для МРХ. При этом мы будем рассматривать не только модули, являющиеся постоянными составляющими любой операционной системы, но и структурные конфигурации, в которые они могут быть объединены.

При этом очень важно, что основные функциональные возможности, приписываемые этим модулям, не зависят от вида мультипрограммирования (с буферизацией или без буферизации ввода-вывода), в условиях которого реализован МРХ.

### **Средства вычислительной системы**

Вычислительные системы, которые мы будем рассматривать, должны состоять из ЦП, оперативной памяти определенного объема, ряда различных периферийных устройств, а также каналов, необходимых для связи устройств с памятью.

Рассмотрим прикладную программу, которая обрабатывается такой системой. Во-первых, эта программа попадает в систему в виде «работы», которая воспринимается с соответствующего носителя одним из устройств ввода (например, устройством ввода с перфокарт). Во-вторых, работа содержит информацию о том, как она должна выполняться (т. е. о шагах ее выполнения), а также, вероятно, о распределении файлов по устройствам, потребностях в оперативной памяти, времени решения, а также указания о том, в какой форме требуется выводить информацию (например, в виде листинга, или в виде перфокарт, или и в том и другом). В-третьих, в работе содержится указания о программах, выполнение которых требуется для выполнения этой работы.

В дальнейшем мы будем полагать, что все аппаратные средства машины, такие, как ЦП, оперативная

память и т. д., а также все те программные средства, которые раньше мы в совокупности называли операционной системой, могут привлекаться для выполнения работы. Так, для того, чтобы определить присутствие работы на устройстве ввода, используется одно из средств операционной системы; для распознавания работы и распределения ЦП и оперативной памяти по рабочим программам этой работы привлекается уже другое ее средство. Во время использования ЦП и оперативной памяти для выполнения команд прикладных программ будет расти количество запросов на устройства ввода-вывода. Понятно также, что эти устройства могут стать доступными только в том случае, если они освобождаются от предыдущей работы, и их функционирование контролируется с помощью программных средств операционной системы, таких, например, как пакет ввода-вывода. Наиболее важным является то, что для выполнения работы привлекаются все необходимые средства, по мере того как в результате ее выполнения генерируются запросы на эти средства.

Выполнение всех таких запросов является, на наш взгляд, одной из основных функций и задач операционной системы.

### **Общие замечания**

При реализации мультипрограммирования центральный процессор распределяется по ряду рабочих программ, которые одновременно находятся в системе. При мультидоступе эти рабочие программы находятся в общей оперативной памяти и последовательно используют ЦП и периферийные устройства, связанные с системой. С этой точки зрения мультипрограммирование можно себе представить как параллельную обработку набора программ, которые одновременно находятся в системе.

Ранее мы уже ссылались на такой набор программ как на мультипрограммный набор или смесь.

Понятно, что для того, чтобы ввести некоторую программу в мультипрограммный набор, требуются вполне определенные программы, которые позволили бы сделать это.

В частности, необходимо вводить в вычислительную систему данные, соответствующие запросам задачи (паспортные данные), т. е. считывать их с промежуточного носителя в оперативную память для того, чтобы могли выполняться соответствующие действия. Таким образом, хотя данные запросов работы и могут существовать вне системы, они приобретают свой смысл и порождают действие, лишь будучи введенными с помощью устройства ввода в оперативную память. Такой ввод является основной функцией ОС, и именно с этого момента начинается фактическая обработка запроса работы.

После того как данные, соответствующие запросу работы, попадают в машину, их необходимо распознать и установить, как система может наилучшим образом удовлетворить требованиям пользователей. Это распознавание и установление наилучшего пути для выполнения работы пользователя является второй основной функцией ОС.

Заметим, что действия системы ограничиваются выполнением лишь тех команд, которые заданы в программе. Отсюда следует, что анализ запроса работы должен в результате завершиться определением тех программ, которые будут выполняться. Однако перед выполнением всякая программа должна быть загружена в память вычислительной системы, и таким образом, третьей функцией ОС является загрузка программ.

Когда программы, соответствующие запросу работы, попадают в память, они тем самым попадают в мультипрограммный набор и с этого момента ждут или пользуются тем или иным средством вычислительной системы. В любой заданный момент в системе могут быть программы, ожидающие обработки на ЦП, а операционная система должна определять, какой из этих программ и когда нужно предоставить такую возможность.

По аналогии с предыдущим случаем может оказаться, что несколько программ из мультипрограммного набора ожидают освобождения одного и того же периферийного устройства, которое связано с оперативной памятью лишь одним каналом. Операционная система и в этом случае должна определять, какая из программ

имеет наибольшее право на использование этого устройства (канала).

Тот модуль операционной системы, на который возложена ответственность за порядок доступа рабочих программ к средствам системы, как раз и является *мультипрограммным исполнителем (монитором)*.

В результате выполнения рабочих программ может возникнуть необходимость в подготовке данных для последующего их вывода на так называемое устройство вывода данных, связанное с данной системой. Примерами таких устройств могут служить дисплеи, графопостроители, АЦПУ, а также перфорационные устройства, как для перфокарт, так и для перфолент, и т. д. Вообще устройство вывода данных представляет собой периферийное устройство, использование которого носит последовательный характер. Это означает, что устройство, занятое одной программой, недоступно другой до тех пор, пока первая его не освободит. Естественно, что такие устройства вывода данных обычно бывают медленно действующими. Данные, которые поступают в устройства вывода, называются *данными для вывода*.

При реализации мультипрограммирования возникают трудности, связанные с тем, что периодически может возникать такая ситуация, когда несколько рабочих программ из смеси одновременно готовят данные для выдачи на одном и том же устройстве. Отсюда вытекает необходимость в механизме, который контролировал бы этот вывод. Подход к организации такого механизма будет рассмотрен ниже. Механизм, предлагаемый нами, должен выполнять две функции. Первая из них заключается в том, чтобы принимать данные для представления от рабочей программы и объединять их в памяти большого объема с данными, подготовленными другими программами. Второй функцией является извлечение данных для вывода и передачи их на соответствующее устройство вывода. Ясно, однако, что при выполнении этой функции для некоторой рабочей программы необходимое устройство будет занято до тех пор, пока не окончится вывод данных для этой программы. В связи с этим, видно, что нужно максимизировать скорость выдачи данных на выбираемых устройствах, что целиком возложено на эту вторую функцию ОС.



В последующих разделах этой главы мы более подробно рассмотрим каждую из вышеупомянутых основных функций ОС, а также дадим некоторые обобщения, касающиеся их динамических взаимосвязей.

## Приемник

Независимо от того, каким образом операционная система оповещается о наличии данных запроса на работу на внешнем устройстве, всякий раз необходимо выполнение функции ввода этих данных в оперативную память, а также некоторых связанных с этим операций. Модуль операционной системы, выполняющий эту функцию, называется *приемником*. Он загружает данные запроса на работу в память, а также осуществляет все необходимые операции по проверке и сборке сообщений. Затем он вызывает следующий модуль операционной системы, который требуется для обработки только что принятых данных запроса на работу.

Запросом на работу может служить последовательность символов, включающая в себя пунктуацию, которая разделяет запрос на единицы информации фиксированного размера (например, информация на перфокарте). Во многих системах, следовательно, от приемника потребовалось бы убрать завершающие пробелы из информации, считанной с карты, и объединить все последующие порции информации с помощью подходящих для этого специальных символов. Одним словом, приемник должен уплотнять поступающую информацию. Заметим, что в большом количестве систем существует возможность совместного ввода запросов на работу через одно и то же входное устройство. В связи с этим может, например, получиться так, что в один пакет приемник считает несколько запросов на работу. Из этого следует, что приемник должен «уметь» определять, где кончается один запрос и начинается другой, чтобы отделить их друг от друга.

Языки, используемые для задания запросов, могут меняться в широком диапазоне, в зависимости от систем, в которых они применяются. В тех случаях, когда такие языки достаточно широки, т. е. когда с их помощью может быть задано большое количество разнообразных элементов запроса (что обеспечивает более

широкие возможности для пользователей), запрос сам по себе увеличивается по объему. В связи с этим возникает проблема хранения запросов, обработанных приемником. Для этого нужно использовать уже не оперативную память, а внешние запоминающие устройства, такие, например, как диски или барабаны. Исходя из этого, приемник должен не только готовить входную информацию в виде, удобном для запоминания, но и снабжать операционную систему ключом к запросу. Последнее означает, что где-то в оперативной памяти, используемой под операционную систему, должна записываться информация, содержащая в себе адрес запроса во внешней памяти, а также идентификатор этого запроса.

Совершенно очевидно, что при движении запросов с устройства ввода в оперативную память, а оттуда во внешнюю память, необходима некая однозначная система идентификации запросов. Далее, вся эта идентификация запросов должна находиться в оперативной памяти, иначе остальная часть ОС не сможет их найти. В качестве такой идентификации может служить запрос ЦП, состоящий из поля запроса и описательного слова запроса, как это было определено в главе 5. В этом случае приемник должен выполнять дополнительную функцию по поиску места в памяти для поля запроса, связанного с поступающим в систему запросом на работу. Поле запроса как минимум представляет собой ячейку памяти, в которую заносится информация, относящаяся к вводимому в систему запросу на работу. Так, например, в этом поле может указываться адрес ячейки, с которой запрос на работу размещается на внешнем носителе.

На практике во многих вычислительных системах язык задания запросов на работу сам по себе до некоторой степени ограничен, и естественно, что длина запроса в связи с этим может быть также ограничена. Таким качеством обладают языки запросов в некоторых системах с неавтономным режимом опроса. Например, запрос на работу в системах с параллельным резервированием схематичен, и множество символов, выражающих его, ограничено.

В этом случае необходимость хранения запросов на работу на внешних запоминающих устройствах

(таких, как диски и барабаны) отпадает. И, наконец, минимально поле запроса может представлять собой ячейку оперативной памяти, в которую заносится запрос на работу, как это уже указывалось ранее.

## Директор

Запрос на работу представляет собой строку символов, которая по своему формату и структуре легко может быть прочитана пользователем. Однако очевидно, что такой язык неудобен для машины, и следовательно, запрос должен быть закодирован так, чтобы его интерпретация внутри машины не вызывала осложнений, т. е. запрос должен быть переведен в удобную для машины форму. Модуль операционной системы, который несет ответственность за эти операции перевода одного формата в другой, называется *директором*.

Кроме того, директор может выполнять также и функцию планировщика, т. е. определять, какие из работ, ожидающих выполнения в вычислительной системе, должны активизироваться вслед за предыдущими. Под активизацией здесь понимается то, что эти работы получают доступ к следующему модулю ОС — загрузчику (он описывается ниже). Таким образом, директор может планировать загрузку для вычислительной системы. И, наконец, директор может осуществлять распределение периферийных устройств и памяти по программам, хотя вообще-то мы будем полагать, что за выполнение этих функций отвечает загрузчик.

В операционной системе функции директора выполняются обычно в комбинации с приемником. Запрос на работу считывается, осуществляется его перевод с языка записи запросов в удобную для машины форму, а после этого он помещается во внешнюю память, откуда впоследствии поступает на анализ, проводимый загрузчиком.

Как уже отмечалось при описании приемника, директор должен запоминать адрес ячейки во внешней памяти, в которой находятся эти просмотренные данные. Впоследствии эта ячейка считывается в оперативную память, и в этих целях как раз очень удобно было бы применять поле запроса, связанное с работой.

Чем шире язык управления заданиями, тем больше вероятность того, что программа, реализующая функцию директора, будет крайне велика. В связи с этим такую программу невыгодно постоянно держать в оперативной памяти, и мы должны были бы рассмотреть процедуру загрузки блоков директора по мере того, как они требуются для обслуживания пакета запросов на работы. При этом будем считать, что время, требуемое на выполнение функций приемника и директора, значительно меньше времени, требуемого на выполнение работы.

В случае же, если время, необходимое для выполнения работ, либо мало, либо соизмеримо со временем выполнения функций приемника и директора, такая частая перезагрузка системных программ может обойтись крайне дорого.

Как уже отмечалось выше, распределение оперативной памяти и устройств по работам может выполняться директором в качестве вспомогательной функции.

Заметим, что в этом случае директор должен распределять и память и устройства, т. е. не следует поручать распределение устройств, скажем, директору, а распределение памяти загрузчику, или доверять такое распределение другим модулям, которые выполняют эти функции асинхронно.

Рассмотрим такой случай, когда распределением устройств занимается директор, а память распределяет некий другой модуль. Если, например, директор уже распределил устройства по работам, а другой модуль через некоторое время пытается «наделить» памятью новые работы, но по каким-либо причинам не может этого сделать в данный момент, возникает задержка, в результате которой устройства, уже закрепленные за этими новыми работами, простаивают. В связи с этим директор сам должен выполнять функцию распределения, которая может расцениваться как вспомогательная. Выполнение ее создает условия для загрузки работы, которая выполняется другим модулем операционной системы. С другой стороны, если распределение вменяется в обязанность загрузчику, то это позволяет повысить эффективность выбора последовательности работ и порядка их инициирования. Такое повышение эффективности выбора достигается за счет анализа по-

следовательности завершения работ, ранее поступивших в систему. Иными словами, если работа заканчивается, закрепленная за ней память освобождается. Загрузчик может оценить параметры работ, поступающих в систему, и выбрать подходящее для загрузки освободившееся место. Тем самым определяется и порядок инициирования, который, как мы уже говорили, эквивалентен загрузке.

## Загрузчик

Для того чтобы программе отвести место в памяти и загрузить ее туда, необходимо, чтобы ее паспортные данные были введены в машину с помощью приемника и проинтерпретированы директором, были представлены в форме запроса на выделение памяти и загрузку. Функция загрузчика заключается в том, чтобы находить требуемые программы, находящиеся на внешних запоминающих устройствах, отводить под них память и, если это необходимо системе, заносить эти программы в отведенные для них участки оперативной памяти.

Периферийные устройства, которые обычно используются прикладными программами, можно разделить на три класса: класс *первичных вводных устройств*, класс *промежуточных устройств* и класс *устройств вывода выходных данных*. В устройства первого класса информация вводится с подготовленных пользователем перфокарт, магнитных лент, дисков и т. д., т. е. ими являются устройства, с помощью которых выполняется первичный ввод программ пользователя с подготовленных им носителей. Таким образом, первичные вводные устройства снабжают прикладную программу исходными данными.

Так как каждый пользователь может подготавливать свою программу на различных носителях, в операционной системе должны быть определены все типы устройств, необходимых для ввода. Сообразно с типом носителя рабочей программе должно присваиваться и периферийное устройство.

Промежуточные устройства используются в качестве внешней памяти, в которую прикладные программы сбрасывают выходную или входную информацию. Эти

устройства, используемые в качестве буферов, играют для пользователя второстепенную роль, так что выбор их может целиком и полностью осуществляться по усмотрению загрузчика.

Устройства вывода данных — это устройства вывода с последовательным принципом действия, обладающие низкими скоростными характеристиками (низким быстродействием). Примером устройства вывода данных может служить АЦПУ. Вообще-то такие устройства не используются в процессе выполнения работы. Они, как правило, выдают готовую информацию пользователю, которая хранится в промежуточном устройстве, таком, например, как диск.

Распределение памяти представляет собой весьма сложную проблему, которая получила достаточное освещение в литературе по программному обеспечению. Хотя методика распределения памяти в значительной степени определяется способом адресации, предусмотренным в конкретной системе, а также механизмом защиты памяти, если таковой имеется, способы распределения памяти можно отнести к одной из двух больших категорий: к категории *смежного* распределения или к категории *несмежного* распределения.

При *смежном* распределении памяти предполагается, что все требуемые ячейки оперативной памяти образуют некий неделимый участок памяти, в котором они смежны физически.

При *несмежном* распределении требуемые ячейки оперативной памяти, расположенные подряд, объединяются в блоки, которые могут находиться в различных участках памяти. Иными словами, при таком распределении необходимые ячейки образуют несколько физически несмежных блоков. Следует заметить, что независимо от способа отведения оперативной памяти под команды программы, какая-то часть памяти обязательно должна распределяться смежно. Это связано с тем, что программа обычно должна иметь рабочее поле памяти, предназначенное для линейно организованных наборов данных, таких, например, как списки, массивы, буфера и т. д., которые, в связи с алгоритмами выборки, обязательно должны находиться в смежных ячейках памяти. Такое требование, однако, не является

обязательным для размещения независимо идентифицируемых программных переменных.

Приемник, директор и загрузчик мы назовем принимающим модулем системы.

### **Монитор (MPX)**

Монитор (мультипрограммный исполнитель, MPX) определяет порядок доступа программ из мультипрограммного набора к средствам вычислительной системы, т. е. центральному процессору, каналам и периферийным устройствам.

Таким образом, MPX выполняет функцию ввода-вывода, а также обработку прерываний, связанных с вводом-выводом. Наряду с этим MPX определяет, какая из конкурирующих рабочих программ должна получить доступ к ЦП.

### **Распределитель**

Распределитель выполняет первую из двух функций, описанных ранее для передачи данных для вывода от рабочей программы в устройство вывода выходных данных. Именно, распределитель должен принимать все записи данных для вывода, подготовленные рабочими программами, и направлять их во внешнюю память большого объема. Организация записей в блоки на этой внешней памяти, а также контроль свободного пространства внешней памяти тоже является функцией распределителя. Кроме того, он может выполнять так же и определенное редактирование данных для вывода, которые он получает от рабочих программ.

### **Диспетчер**

Диспетчер выполняет вторую функцию модуля вывода системы и тесно связан с работой распределителя. В его задачи входит считывание данных для вывода из внешней памяти и передача этих данных в соответствующие устройства для вывода, при которой обеспечивалась бы максимальная пропускная скорость этих устройств.

Данная работа не содержит подробного описания такой методики хранения информации, которая была бы наилучшим образом приспособлена к совместной работе распределителя и диспетчера, так как она зависит главным образом от периферийного оборудования, имеющегося в вычислительной системе.

Мы ограничимся лишь общими рассуждениями об этих методах, которые исходят из того, что в системе большое количество пользователей стремится получить доступ к устройствам вывода выходных данных. С нашей точки зрения, доступ к устройству вывода эквивалентен доступу к распределителю, который передает данные для вывода из разных источников в одну общую внешнюю память.

Ясно, что внешняя память, используемая распределителем и диспетчером, должна быть памятью с произвольным доступом. Более того, распределителю так нужно организовать данные для вывода, предназначенные для выдачи, например, на АЦПУ, чтобы они в памяти с произвольным доступом располагались в логической последовательности. Это требуется для того, чтобы диспетчер при передаче данных для вывода, связанных с выбранной прикладной программой, мог находить их в требуемой последовательности. Кроме того, организация логической последовательности записей должна быть достаточно эффективной для того, чтобы программе диспетчера не приходилось производить чрезмерно сложные вычисления адреса каждой последующей записи. Если эти обширные вычисления все же должны производиться, будет теряться не только полезное время ЦП, необходимое для выполнения прикладных программ, но и может появиться задержка, равная результирующему старт/стопному времени устройства вывода выходных данных, а вследствие ее появления может ухудшиться и пропускная способность этого устройства.

Для того чтобы минимизировать общее время ЦП, требуемое для выполнения функции диспетчера, нужно минимизировать время, необходимое для выполнения его подфункций, а также сокращать число этих подфункций. В частности, все функции редактирования, а также основные функции передачи данных нужно отдать распределителю.



Ниже перечисляются основные модули (6 модулей), необходимые для реализации мультипрограммной операционной системы, и функции, которые они должны выполнять.

### **П р и е м н и к**

Ввод запроса на работу (ввод паспортных данных).  
Выполнение программы ввода.  
Системное представление входной информации.  
Формирование блоков информации.

### **Д и р е к т о р**

Интерпретация запроса на работу.  
Преобразование символов.  
Приведение к требуемому формату.  
Распознавание работ.  
Контроль блоков информации.

### **З а г р у з ч и к**

Загрузка программы.  
Распределение памяти.  
Перераспределение памяти.  
Связь модулей.  
Перезагрузка модулей системы.

### **М о н и т о р**

Распределение ЦП.  
Распределение устройств ввода-вывода.  
Защита памяти.  
Обработка прерываний.  
Организация системы приоритетов на каналы.  
Организация системы приоритетов для устройств с произвольным доступом.  
Управление буферизацией.  
Управление обращениями к файлам.  
Организация управления устройствами ввода-вывода.

### **Р а с п р е д е л и т е л ь**

Обработка выходной информации для вывода.  
Преобразование символов.  
Приведение к необходимому формату.  
Управление внешней памятью.

## Д и с п е т ч е р

Управление выходными данными для вывода.

Оптимизация пропускной способности устройств.

### Взаимосвязи между модулями ОС

Каждый из модулей операционной системы выполняет определенные операции над данными запросов на работу и рабочими программами, связанными с этими данными, а также над данными для вывода.

Вообще говоря, работа начинает свое «движение» через систему с приемника, затем «проходит» через директор и загрузчик и в качестве текущего обрабатываемого запроса «проходит» через исполнитель, под управлением которого она завершается.

В процессе выполнения работы вырабатываются данные для вывода, которые проходят через распределитель и диспетчер в устройства вывода выходных данных.

На диаграмме, показанной на рис. 14, приближенно отражено движение работы через систему.

Запрос на работу через приемник может попасть либо в память, оперативную или внешнюю, в зависимости от конкретной системы, либо директору предписывается обработать этот запрос. В последнем случае предписание на обработку запроса, т. е. на вызов директора, помещается в очередь DIRQ, связанную с директором. Количество вызовов, которые могут быть поставлены в эту очередь и одновременно находиться в ней, также зависит от конкретной системы.

Независимо от того, как запрограммирован директор, при его выполнении должны осуществляться три основных действия. Во-первых, директор осуществляет доступ к данным запроса на работу, занесенным в память приемником, после чего он анализирует эти данные.

Далее проанализированные данные трансформируются директором в данные для загрузки, форма которых удобна загрузчику. После этого данные для загрузки либо остаются в оперативной памяти (может быть, в другом ее участке), либо заносятся во внешнюю память для последующего их использования загрузчиком.

И, наконец, поскольку данные для загрузки уже подготовлены и помещены в память, делается вызов

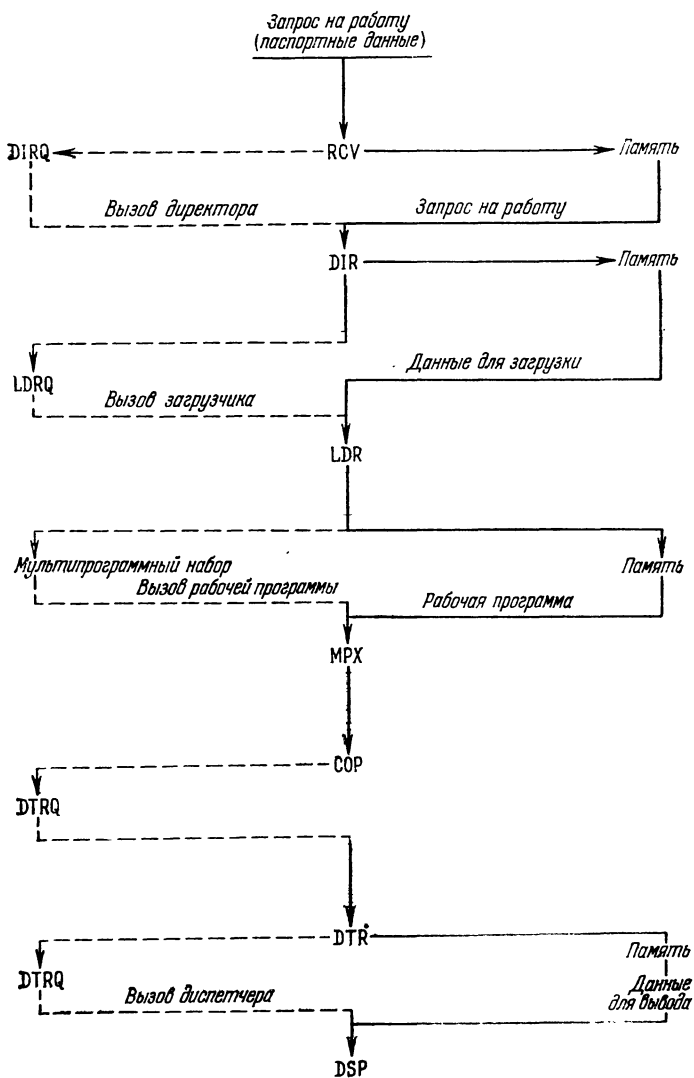


Рис. 14. Движение работы через систему.

загрузчика. Этот вызов помещается в очередь на загрузчик LDRQ (см. рис. 14).

Так же как и директор, загрузчик при своем выполнении осуществляет три основных действия.

Во-первых, данные для загрузки считываются из отведенной для них памяти, и по ним распознаются программы, загружаемые с внешнего носителя, на котором они хранятся. После того как определено, какие программы должны быть считаны в оперативную память (и, кроме того, произведено распределение оперативной памяти и периферийных устройств), загрузчик находит эти программы во внешней памяти, загружает их в оперативную память и переписывает указанные программы в отведенное для них место в оперативной памяти. Когда загрузка завершена, рабочая программа, соответствующая запросу на работу, находится в оперативной памяти, и в связи с этим загрузчик должен сообщить исполнителю о том, что мультипрограммный набор пополнился новой программой. Для этого загрузчиком формируется специальная информация, которая впоследствии анализируется исполнителем.

Основная задача монитора заключается в определении того, какие программы из мультипрограммного набора должны получать доступ к средствам системы при условии, что все эти программы одновременно претендуют на указанные средства. В частности, при проведении такого анализа одна из таких программ получает доступ к ЦП.

Такая программа является для MPX текущей обрабатываемой (COP). Кроме того, MPX во время своего выполнения, т. е. во время поисков очередной COP, сам может рассматриваться как COP.

Как показано на рис. 14, текущая обрабатываемая программа во время своего выполнения осуществляет два действия, имеющие отношение к операционной системе. Во-первых, она вырабатывает данные для вывода и, во-вторых, формирует вызов распределителя для обработки этих данных. Из рисунка видно, что вызов распределителя попадает в очередь вызовов DTRQ.

Основные функции распределителя заключаются в принятии данных для вывода от COP и передаче этой информации в специально отведенные участки внешней

памяти большого объема. Для каждого из устройств вывода выходных данных, которые контролируются распределителем и диспетчером, распределитель должен каждый раз связывать выходные данные с соответствующей группой данных, снабженной идентификатором программы.

Такая проблема управления файлами выходит за рамки рассматриваемой нами функции распределителя мультипрограммной операционной системы, и сама по себе в данной работе не рассматривается.

Последним модулем в операционной системе является диспетчер. В его функции входит управление устройствами вывода выходных данных, которое обеспечивало бы максимальную пропускную способность этих устройств. Наряду с этим диспетчер должен устанавливать и контролировать последовательность выдачи выходных данных. При этом необходимо должны быть предприняты меры по предотвращению смешивания выходных данных для разных программ, которые предназначены для выдачи на одном и том же устройстве. Отсюда вытекает, что диспетчер, в основном, работает с внешней памятью и устройствами вывода выходных данных. Он блоками считывает данные из памяти и передает отдельные записи из этих блоков в периферийные устройства. Основной проблемой в организации диспетчера является эффективная выборка следующей группы данных для выдачи на заданном устройстве вывода выходных данных. В данной работе мы не будем рассматривать подробно методы, с помощью которых эта проблема разрешается. Нас будет интересовать лишь тот факт, что такая выборка выполняется и что она должна быть достаточно эффективной.

## **Запросы**

При выполнении функций основных модулей системы, а именно: модуля принятия, исполнителя и модуля вывода выходных данных, запрос на работу будет как бы проходить через вычислительную систему. Как мы уже говорили в главе 5, процесс выполнения самой работы может быть отражен в некотором досье этой работы, названном нами запросом ЦП. Так как каждая функ-

ция операционной системы связана с программой, которая ее выполняет, мы можем присвоить запросы и этим функциям (т. е. программам). Таким образом, запрос, связанный с модулем ввода системы, порождает запрос на работу для новой работы и направляет этот запрос в мультипрограммный набор. Именно с этим запросом и работают исполнитель и модуль вывода выходных данных для того, чтобы осуществить это прохождение работы через систему. В последующих параграфах основные функции операционной системы будут описаны на языке запросов.

### **Модуль ввода**

О том, что на внешнем вводном устройстве находится запрос на работу, система оповещается либо с помощью анализа состояния внешнего переключателя, либо за счет выполнения внутреннего прерывания.

В этот момент происходит первое знакомство системы с новой работой, и при этом для нее формируется запрос, состоящий из поля запроса и связанного с ним слова запроса. Поле запроса первоначально содержит информацию, устанавливающую связь между работой и устройством, на которое поступил запрос на работу. Это поле формируется в отведенном месте оперативной памяти.

Затем слово запроса, используемое как указатель местонахождения поля запроса в оперативной памяти, помещается в очередь RCVQ (receiver queue), которая анализируется приемником (т. е. слово запроса попадает в одну из ячеек оперативной памяти, отведенных под эту очередь и просматриваемых приемником).

Благодаря такой организации запроса на работу, приемник во время своего выполнения, проанализировав слово запроса, получает доступ к полю запроса, которое, в свою очередь, снабжает его информацией, необходимой для считывания данных запроса на работу (паспортных данных) с соответствующего периферийного устройства. После того как данные запроса на работу считаны в оперативную память и над ними проведены все необходимые операции (например, формирование блоков информации из них и т. д.), они, в зависимости от конкретной системы, либо заносятся

в отведенное место в оперативной памяти, либо направляются на хранение во внешнюю память.

Заметим, что данные запроса на работу могут записываться в поле запроса и храниться вместе с ним в оперативной памяти. Пример мы приведем ниже.

В любом случае последней операцией, которая выполняется приемником, является засылка слова запроса в одну из ячеек ОП, отведенных под очередь DIRQ, благодаря чему директор может работать с этим словом.

При выполнении директора связующим звеном между ним и полем запроса служит все то же слово запроса. В свою очередь, поле запроса снабжает директора информацией, используя которую, он может найти данные запроса на работу и манипулировать ими. Найдя эти данные, директор может, проанализировав их, определить, какие файлы необходимы для выполнения работы, какие для этого требуются устройства, а также какие программы требуются загрузить, чтобы исполнить этот запрос на работу. Кроме того, для программ, которые требуется загрузить, он может определить их местоположение на внешних носителях, необходимое количество памяти, которое нужно для них отвести, а также тип ее организации при распределении (смежная, несмежная ОП). Вся эта информация может заноситься в поле запроса или временно храниться в ячейках, адреса которых зафиксированы в поле запроса. Последней операцией, которую выполняет директор, является занесение слова запроса в одну (несколько) из ячеек, отведенных под очередь LDRQ. Иными словами, запрос ставится в очередь на загрузчик. Благодаря этому загрузчик может работать с этим запросом.

Как и в предыдущем случае, связующим звеном между загрузчиком и полем запроса служит слово запроса. Так как после выполнения директора в поле запроса содержится вся необходимая информация, относящаяся к загрузке программ, а также, возможно, и информация, касающаяся распределения памяти и устройств, то загрузчик может сразу же приступить к выполнению своих функций, т. е. выполнить распределение и загрузку. Когда проведено распределение памяти и устройств, а также осуществлена загрузка программ в память, информация о распределении и начальных адресах за-

рузки заносится в поле запроса. Теперь, когда рабочие программы загружены в память, запрос становится членом мультипрограммного набора. Следовательно, последнее, что должен сделать загрузчик,— это занести слово запроса в список слов запросов.

### **Исполнительный модуль**

Как уже говорилось ранее, монитор выполняет в основном две функции. Во-первых, он контролирует доступ к ЦП, а, во-вторых, определяет порядок доступа к средствам ввода-вывода системы. В связи с этим мы рассмотрим те операции, которые выполняет МРХ над запросом пользователя при реализации этих двух функций.

При выборе следующей программы для очередного выполнения, МРХ фактически просматривает список слов запросов (этот список и представляет собой мультипрограммный набор) и выбирает одно из этих слов.

Сами по себе критерии выбора мы рассмотрим позднее, а сейчас нас интересует лишь тот факт, что, независимо от того, какой критерий используется, выбирается лишь одно слово запроса. После того, как сделан выбор, МРХ анализирует соответствующее поле запроса, к которому он получил доступ, благодаря адресу, указанному в слове запроса. В поле запроса он находит адрес команды, с которой нужно продолжить выполнение программы, представленной в мультипрограммном наборе данным словом запроса. Кроме адреса команды программы, в поле запроса хранится также информация, содержавшаяся в индекс-регистрах к моменту прерывания. Отсюда вытекает, что МРХ должен записать эту информацию в соответствующие индекс-регистры (т. е. восстановить их).

После того как выполнены эти операции, МРХ готов к передаче управления на выбранную программу, для того чтобы ее выполнение было продолжено. Однако для осуществления этой передачи управления нужно осуществить еще одну манипуляцию со словом запроса. Это слово нужно занести в специальную ячейку памяти, чтобы отличать его от всех других слов мультипрограммного набора. В главе 5 мы называли его сло-



вом текущего обрабатываемого запроса или СОТ, а ячейку, в которую оно заносится, подполем СОТ поля ЦП. Таким образом, когда обрабатывается программа, относящаяся к этой работе, слово запроса для нее находится в специальной ячейке памяти СОТ. И, наоборот, программа, слово запроса которой находится в СОТ, является текущей обрабатываемой (СОР).

Работа исполнителя для мультипрограммирования с небуферируемым запросом на ввод-вывод уже рассматривалась нами в главах 4 и 5. В этих главах мы обсуждали выполнение операций ввода-вывода и роль, которую играет МРХ в их реализации. Напомним, что по запросу ЦП можно было судить об использовании средств ввода-вывода для той или иной работы (т. е. при анализе его подполей можно было однозначно определить, использует в данный момент данная работа средства ввода-вывода или нет). Однако это справедливо лишь в случае мультипрограммирования с небуферируемым запросом на ввод-вывод, где использование ЦП для одной и той же задачи не может быть совмещено во времени с использованием средств ввода-вывода.

В случае же, когда реализована буферизация, одним запросом ЦП не обойтись, и возникает необходимость в запросе на ввод-вывод. Использование таких запросов в условиях мультипрограммирования буферируемого запроса на ввод-вывод мы рассмотрим позднее, в главе 8, а сейчас ограничимся сказанным выше и будем полагать, что МРХ при выполнении запросов на ввод-вывод, которые порождаются запросом ЦП, должен манипулировать запросом ЦП и осуществлять его запоминание.

### **Модуль вывода выходной информации**

Чтобы проследить дальнейшее движение запроса через операционную систему, рассмотрим работу распределителя. Напомним, что этот модуль системы используется в тех случаях, когда рабочая программа генерирует данные для вывода, т. е. запись, которая состоит из данных, предназначенных для выдачи на внешние

устройства, но еще не приведенных к соответствующему виду.

Так как генерация таких данных на всем протяжении выполнения программы связана со словом запроса ЦП, то распределитель имеет доступ ко всей необходимой информации, имеющей отношение к данным для вывода, а также к информации, связанной с размещением в памяти большого объема предыдущих порций данных, которые выработала ранее данная программа. Такая информация находится, конечно, в поле запроса или же в тех областях памяти, адреса которых хранятся в этом поле. Связь с данными происходит именно через поле запроса. Следовательно, распределитель, просмотрев поле, может найти адрес ячейки, в которой размещаются данные для вывода, а также начальный адрес поля памяти, предназначенной для новых данных.

Предполагается, что после того, как распределитель получит последнюю запись данных для вывода от рабочей программы и запишет их в соответствующее место внешней памяти, будут выполнены операции по закрытию файла.

Это означает, что будут сгенерированы все необходимые признаки конца блока данных, относящихся к этой программе. Иными словами, информация, выданная программой, приобретает некий законченный вид и будет передаваться диспетчером из внешней памяти на устройство выходных данных, которое закреплено за этой программой. Для того, чтобы такая передача была выполнена, распределитель направляет диспетчеру соответствующую информацию (например, копию слова запроса, относящегося к этой программе). Таким образом, когда вызывается диспетчер, в его распоряжении уже имеется информация, эквивалентная слову запроса ЦП, которая свидетельствует о наличии данных для вывода, сгенерированных рабочей программой. Это слово запроса служит для диспетчера ключом к информации о самих данных, находящихся в памяти большого объема.

Использование слова запроса и информации, находящейся в поле запроса, зависит уже от того, как построен диспетчер (от его конкретной логической структуры).

Заметим, что запрос ЦП для рабочей программы можно было бы давно выбросить из системы, так как выполнена соответствующая ему рабочая программа. Это можно было бы сделать даже, если одна или несколько записей с данными для вывода, относящимися к этой программе, еще не обработаны диспетчером. Однако запрос ЦП для работы устраняется из системы только после того, как будут выданы все данные, связанные с этой работой, и тем самым эта работа будет завершена.

### Общие замечания

Рассматривая основные модули операционной системы и выполняемые ими функции, мы, главным образом, ограничивались их статическими характеристиками. Главная мысль этой работы заключается в том, что любая операционная система должна содержать приведенные выше основные модули, и всегда ее работа может рассматриваться с точки зрения функционирования этих модулей. Наряду с этим каждая операционная система должна фиксировать данные, соответствующие работе, выполняемой для заданного пользователя, и, по нашему глубокому убеждению, запрос может наилучшим образом использоваться в этих целях. Благодаря такой организации (т. е. слово запроса + поле запроса) запрос весьма удобен при поиске и фиксации данных, находящихся в разных местах памяти, а также более чем пригоден для регистрации событий, происходящих в системе.

Итак, подводя некоторый итог всему сказанному в предыдущих главах, можно сделать вывод, что мы продвинулись в изучении операционной системы не дальше ее основных функций, выработав при этом некий базовый информационный механизм, который позволяет нам фиксировать состояние работы на различных этапах ее выполнения в системе. Однако, если мы перейдем теперь от статических характеристик системы к ее динамическим характеристикам, мы сможем сделать более важные выводы, которые будут уже иметь фундаментальное значение. В связи с этим последующие разделы этой книги будут посвящены рассмотрению общих приемов формального анализа операционной системы.

## Динамические взаимосвязи

При рассмотрении основных модулей операционной системы мы почти не касались взаимосвязей между ними, было сказано лишь то, что такие связи существуют и что связующим звеном в конечном итоге является запрос.

Запрос так или иначе попадает в мультипрограммный набор, где с этого момента он начинает представлять работу пользователя как рабочую программу, готовую к выполнению.

Хотя различные модули операционной системы и манипулируют запросами пользователей, вводя их в систему, занося в мультипрограммный набор, выполняя над ними различного рода операции и используя для вывода выходных данных, сами эти модули являются программами и, следовательно, в системе могут быть представлены запросами. При реализации такого подхода до некоторой степени упрощается организация совместной работы модулей. Так, например, для того, чтобы выполнялся директор, МРХ должен лишь выбрать его запрос в качестве текущего обрабатываемого.

А теперь воспользуемся вышеприведенным свойством запросов для того, чтобы исследовать динамические взаимосвязи, существующие между модулями операционной системы.

Предположим, например, что при поступлении запроса на работу на внешний носитель происходит прерывание СОТ, и МРХ перемещает слово запроса для приемника из стандартной ячейки операционной памяти R1 в стандартную ячейку R2. Ячейка R1 обнуляется, и выполнение прерванной программы возобновляется. Если задержка в выполнении рабочей программы вызывает необходимость возврата к МРХ для перераспределения ЦП, исполнитель сразу же анализирует содержимое ячейки R2 на наличие слова запроса, связанного с приемником.

Если оно в R2 имеется, МРХ делает его текущим обрабатываемым (СОТ) и, таким образом, обеспечивает выполнение программы приемника, в результате которого в систему попадают данные, связанные с ожидающим ввода запросом на работу. Сказанное выше иллюстрирует блок-схема на рис. 15.

Предположим теперь, что программа приемника немедленно передает управление директору, как только данные запроса на работу введены в память и из них сформированы блоки информации. Для этого приемник должен удалить свое собственное слово запроса из ячейки COT и поместить его в R1. Кроме того, ему нужно переслать слово запроса, принадлежащее директору, из некоторой стандартной ячейки, скажем D1, в ячейку COT, в результате чего директор станет текущей обрабатываемой программой (см. рис. 16). Аналогичным образом, после окончания своей работы, директор манипулирует запросом загрузчика и делает его текущим обрабатываемым.

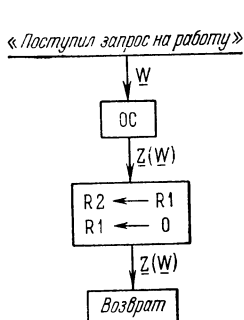


Рис. 15. Активизация приемника.

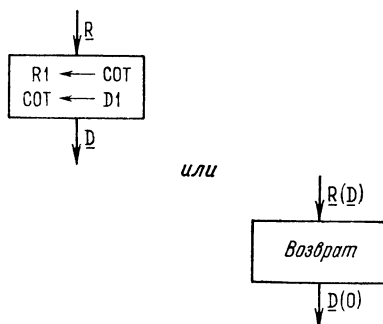


Рис. 16. Завершение работы приемника.

После выполнения своих функций загрузчик перешлет свое собственное слово запроса обратно в стандартную ячейку (например, L1) и передаст управление на MPX, который, в свою очередь, займется выбором следующего COT из мультипрограммного набора.

Предположим, что сработало первое звено в только что построенной нами системе, т. е. MPX, после того как прервано выполнение рабочей программы, передал слово запроса, связанное с приемником, из R1 в R2, обнулil R1 и обеспечил возобновление выполнения прерванной программы.

Предположим теперь, что сразу же после поступления одного запроса на работу происходит следующее прерывание, сигнализирующее о наличии нового запроса на устройстве ввода. Нулевое содержимое R1

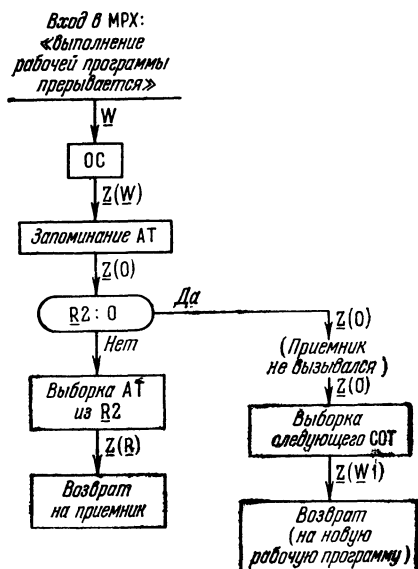
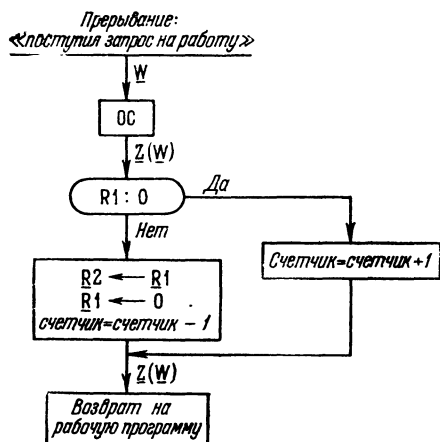


Рис. 17. Работа приемника в мультипрограммном наборе с запросами ОС.

будет указывать на то, что приемник, директор и загрузчик (т. е. принимающий модуль операционной системы) занимаются обработкой только что полученного запроса на работу (точнее говоря, управлением сформированного внутри системы аналога этого запроса). Поскольку в рассматриваемой нами системе принимающий модуль не может параллельно обрабатывать несколько запросов на работу (наложим такое ограничение на нашу систему), то формирование аналога для нового запроса должно быть отложено. Зарегистрировать приход нового запроса на работу можно, связав ячейку R1 со счетчиком, содержимое которого увеличивается на единицу всякий раз, когда приходит новый запрос на работу, и уменьшается (также на единицу) в случае, если слово запроса пересылается из R1 в R2 (см. рис. 17).

Элементарный вариант системы, описанный выше, дает представление о непосредственных взаимосвязях модулей операционной системы. Эти модули работают последовательно друг за другом, и выполняются в промежутках между выполнением рабочей программы.

Приведенные выше взаимосвязи модулей в системе являются простыми и в общем-то не требуют дальнейших пояснений.

А сейчас мы вкратце рассмотрим более сложный вариант системы с точки зрения взаимодействия ее модулей, а более подробное описание такой ОС отложим до следующей главы.

Итак, предположим, что мультипрограммный набор содержит не только слова запросов для рабочих программ, но, и, кроме того, слова запросов для каждого модуля операционной системы (модуля ввода). Иными словами, в этот набор добавляются еще слова запросов для приемника, директора и загрузчика. Это означает, что критерии, которые используются МРХ при выборе СОТ из слов запросов рабочих программ, будут распространяться и на слова запросов системных программ, находящихся в этом мультипрограммном наборе. В результате выбора текущим обрабатываемым запросом может стать как **системный**, так и **рабочий** запрос, все зависит от сложившейся ситуации и выполняющихся критериев, общих для всех запросов. Отсюда вытекает, что уровень мультипрограммирования, достигнутый

ОС при обработке рабочих программ, будет в этом случае характерен и для выполнения системных программ, попавших в мультипрограммный набор. В результате получаем мультипрограммную операционную систему.

Забегая немного вперед, скажем, что в результате добавления слов запросов для распределителя и диспетчера к уже добавленным нами системным словам запроса в мультипрограммном наборе получается операционная система с новыми свойствами.



## ГЛАВА 7

# ОПЕРАЦИОННАЯ СИСТЕМА ДЛЯ НЕБУФЕРИРУЕМЫХ РАБОТ

### Введение

В главе 4 мы показали, что центральный модуль операционной системы — *монитор*, полностью определяется при условии, если все программы, выполняемые под его управлением, осуществляют все необходимые операции ввода-вывода без буферизации.

При этом же условии были определены три взаимоисключающих состояния рабочей программы (т. е. взаимоисключающие в том смысле, что программа в один и тот же момент своего выполнения может находиться в одном и только в одном состоянии). При этом условии логическая структура монитора определяется сравнительно легко.

В главе 5 было введено понятие запроса и рассмотрены некоторые возможности запросов при анализе операционной системы, на примере построения схемы запросов МРХ для ОС (случай мультипрограммирования с небуферируемым запросом на ввод-вывод).

В настоящей главе мы воспользуемся методом запросов для построения всей ОС полностью, включая модули, рассмотренные в главе 6.

До настоящего момента при описании операционной системы, мы, как правило, подробно останавливались лишь на определениях данных и модулей, которые имеют к ней отношение, а также, в основном, на проблеме структуры системы в целом. Подробному анализу подвергся только МРХ для мультипрограммирования

с небуферируемым запросом на ввод-вывод. Следует отметить, что, хотя на логическую структуру исполнителя и наложены определенного рода ограничения, конфигурация операционной системы и ее общая производительность зависят от организации модуля ввода и модуля вывода. При разработке этих модулей, несмотря на то, что они должны осуществлять все необходимые операции ввода-вывода без буферизации, существует некая свобода действий. Иными словами, не имеет значения, как построены модули, лишь бы они удовлетворяли предъявляемым требованиям, а именно: выполняли бы операции ввода-вывода без буферизации.

Таким образом, хотя МРХ в ОС и является жестко зафиксированным модулем, вся система в целом ориентируется каждый раз на конкретное оборудование. В этой главе как раз и рассматривается пример построения ОС для конкретного случая (т. е. берется некоторая конкретная вычислительная система, выявляются требования, предъявляемые оборудованием к ОС, и строится некоторая ОС, удовлетворяющая этим требованиям). Операционная система, которую мы построим, будет полностью мультипрограммируемой, т. е. все программы модуля ввода и модуля вывода войдут в мультипрограммный набор. При анализе этой системы мы воспользуемся методом запросов.

### **Постановка задачи**

Для того чтобы приступить к разработке операционной системы, нужно прежде всего выявить требования, предъявляемые к ней в целом. Ниже перечисляются требования к системе, которая будет рассматриваться в нашем примере.

Запросы на работу должны поступать в ОС через устройство ввода с перфокарт. Входные данные для рабочих программ должны поступать с быстродействующих устройств, таких, например, как магнитные ленты или диски. В случае же, если входные данные поступают с перфокарт, система должна, следовательно, обеспечивать выполнение функции переноса информации с перфокарт во внешнюю память.

Предполагается, что скорость компиляции достаточно высока для того, чтобы не прерывать процесса

обработки рабочей программы (т. е. ее компиляцию). Иными словами, программа-компилятор успевает выполнить свои функции за один раз, и повторно вызывать ее для обработки одной и той же программы не требуется. С другой стороны, частота обращений к этой программе не так уж велика, чтобы постоянно держать ее в оперативной памяти. Следовательно, компилятор мы будем рассматривать как рабочую программу, которая загружается в память при соответствующем обращении и может использоваться повторно в случае, если дополнительные запросы приходят тогда, когда она является резидентной.

То же самое в равной мере относится и к программе выполняющей перенос входных данных из оперативной во внешнюю память. Она должна быть резидентна в ОП только на время своего выполнения.

И, наконец, будем предполагать, что на АЦПУ будет выводиться большой объем информации. Для этих целей в вычислительную систему будут входить два неавтономных АЦПУ. Задача операционной системы заключается в том, чтобы обеспечить максимальную пропускную способность этих устройств за счет соответствующей организации мультидоступа к ним.

### **Конфигурация аппаратного обеспечения**

В систему, которую мы будем в дальнейшем рассматривать, как минимум должны входить в ЦП и оперативная память, связанная с двумя устройствами ввода с перфокарт, двумя АЦПУ и диском. Эти устройства будут так или иначе использоваться при выполнении различных функций операционной системы. Кроме того, в эту систему должны входить магнитофоны и внешние запоминающие устройства большого объема, которые будут использоваться большими программами.

В последующих параграфах мы дадим обзор некоторых модулей операционной системы, а затем проанализируем их на основе метода запросов.

#### **Модуль ввода**

Одно из устройств ввода с перфокарт (устройство 1) мы отведем под прием запросов на работу. Будем предполагать, что при поступлении нового запроса на это

устройство будет происходить сигнальное прерывание, обрабатываемое МРХ. Это прерывание оповестит операционную систему о вновь поступившем запросе на работу. Опознав, что это прерывание связано с устройством 1, МРХ будет формировать новый внутренний запрос, который отныне будет представлять эту работу в системе. Этот запрос будет состоять из поля запроса и слова запроса, описанных в главе 5. Для поля запроса МРХ отведем необходимое место в памяти и будем заносить туда всю информацию, которая будет выработана в течение обработки запроса модулем ввода. Что касается слова запроса, то в нем будет находиться имя поля запроса. Имя поля запроса — это не что иное, как начальный адрес только что сформированного поля запроса в ОП. После того, как МРХ выполнил вышеперечисленные действия, он ставит новое слово запроса в очередь, состоящую из слов запросов, пришедших ранее и ожидающих обработки приемником. Эту очередь мы будем называть очередью на приемник или RCVQ (от английского receiver queue). После этого МРХ определяет, в каком состоянии (активном или пассивном) находится приемник, и, если это требуется, активизирует его, ставя связанное с ним слово запроса R в очередь готовности AQ.

Когда приемник активизирован, он будет производить выбор соответствующего слова запроса из RCVQ. После того как это слово выбрано, приемник считывает данные запроса на работу с устройства 1 в то место памяти, которое отведено для поля этого запроса. Таким образом, поле запроса используется в первую очередь для хранения данных запроса на работу, поступающих с устройства ввода.

После этого приемник ставит слово обработанного им запроса в очередь на директор или DIRQ (director queue). Последней функцией, которую выполняет приемник, является активизация директора, если он не работает в данный момент.

Директор начинает свою работу с выборки соответствующего слова запроса из DIRQ. Это слово запроса содержит в себе адрес связанного с ним поля, в котором находятся данные запроса на работу, занесенные приемником. Затем директор интерпретирует эти данные, определяя, что требуется сделать: выполнить ли функцию

переноса данных на внешний носитель, оттранслировать программу или начать выполнение обычной рабочей программы.

Если, например, требуется выполнить функцию переноса данных во внешнюю память, то системная программа, реализующая эту функцию, выполняется так, как будто она является обычной рабочей.

Если в запросе на работу указано, что необходима компиляция, то слово данного запроса связывается с запросом для программы-компилятора, и в совокупности эти слова запросов образуют некую комбинацию, проанализировав которую, операционная система (в частности, МРХ) будет инициировать выполнение программы-компилятора. Метод организации этой связи слов запросов мы подробно рассмотрим несколько позже.

Заметим, что при инициировании компиляции у директора есть два пути, сообразуясь с которыми, он принимает то или иное «решение». В частности, если в данный момент программа-компилятор загружена в ОП (и поскольку программа повторно входима), то ее выполнение незамедлительно может быть инициировано для данного запроса простым занесением комбинированного слова запроса в очередь готовности.

В случае же, если компилятора в данный момент нет в ОП, этот комбинированный запрос будет рассматриваться как запрос обычной рабочей программы.

Директор, интерпретируя данные запроса на работу, должен определить, какие устройства требуются рабочей программе, и после этого «выяснить», какие из этих устройств можно ей приписать. Дело в том, что все или часть требуемых устройств могут быть уже приписаны предыдущим программам и освободятся только по их выполнении. Если все требуемые устройства уже приписаны другим программам или же их просто нет (так, например, не установлена необходимая бобина на магнитофоне, отсутствует запрошенный диск и т. д.), то слово данного запроса помещается в список директора и делается запись о требуемых устройствах (может также выдаваться сообщение.) Этот запрос будет стоять в списке директора до тех пор, пока другие программы не освободят необходимые устройства.

Когда эти устройства, наконец, приписываются данному запросу, его слово помещается в очередь на загрузчик или LDRQ (loader's queue). После этого директор определяет, в каком состоянии находится загрузчик (в пассивном или активном), и при необходимости активизирует его.

Загрузчик начинает свою работу с анализа LDRQ и выборки соответствующего слова запроса. В той ОС, которую мы рассматриваем, выбор проводится по критерию размера требуемой памяти. Запрос, для которого необходима наибольшая память (среди остальных запросов), будет обладать наинизшим приоритетом. Далее загрузчик определяет: можно ли данному запросу отвести тот объем памяти, который ему требуется, и если нельзя — возвращает этот запрос обратно в LDRQ. В последнем случае он прекращает свою работу и лишь по завершении какой-либо программы вновь активизируется монитором (MPX). Таким образом, выполнение программ, запрашивающих большой объем памяти, откладывается на неопределенное время. Это, в частности, относится и к рассматриваемой нами системе, в которой, как мы предполагаем, компилятор будет использоваться достаточно часто, а он, что весьма вероятно, будет одной из самых больших программ, использующих ее.

### **Модуль вывода**

К вычислительной системе подключено два устройства печати и системный диск. Таким образом, для эффективного вывода необходимо иметь в системе программу-планировщик и программу-диспетчер.

Распределитель принимает заказы от программ, работающих в мультипрограммном режиме. Эти заказы означают запрос вывода на печать. Данные, подлежащие выводу, располагаются в буфере, находящемся на системном диске операционной системы. Планировщик активизируется при помощи запроса на ввод-вывод, направляемого в MPX. MPX помещает запрос соответствующей программы, требующей вывода, в очередь планировщика. В дальнейшем эту очередь мы будем обозначать DTRQ. Затем MPX активизирует распределитель, если он еще не находится в работе.

Распределитель выбирает запрос из очереди DTRQ, помещает запись, предназначенную для печати, в блок таких записей для данной рабочей программы и по заполнении этого блока переписывает его в буфер для печати, находящийся на системном диске. Независимо от того, заполняет ли данная запись блок до конца или нет, задача, потребовавшая вывод на печать, помещается опять в очередь задач, ожидающих решения.

Программа-диспетчер обладает свойством повторной входимости и связана с двумя блоками выдачи сообщений — по одному на каждое устройство печати. Эти блоки активизируются при окончании программы пользователя. В этом случае запрос на сообщение об окончании рабочей программы помещается в очередь диспетчера DSPQ. Соответствующий блок MPX определяет, имеется ли незанятый работой блок выдачи сообщений. Если да, то данный блок активизируется.

В начале работы программа-диспетчер выбирает запрос из очереди DSPQ. Затем она помещает первый блок данных, подлежащих выдаче по данному запросу, в буфер печати на диске и переносит этот блок в оперативную память. В то время, пока первый блок выдается на печать, следующий блок записей переносится из буфера в оперативную память. Этот цикл повторяется до тех пор, пока не будет выдана на печать вся информация по данному запросу. После этого работа по данному запросу считается законченной, и запрос уничтожается.

### **Монитор (MPX)**

Монитор для этой операционной системы по существу такой же, как и описанный в 4 и 5 главах. Так как выполнение программ приема, выдачи и рабочих программ строится на основе небуферизируемого ввода-вывода, то основные изменения в MPX касаются, в основном, директора, загрузчика, распределителя и диспетчера.

В программе-директоре может стать необходимым исключить передачу запросов загрузчику из-за невозможности доступа к периферийным устройствам. Следовательно, функция опроса периферийных устройств, освобождаемых оканчивающейся программой, возла-

гается на блок окончания программ пользователя, входящий в состав МРХ. Запросы, отложенные директором до освобождения периферийных устройств, снова помещаются этим же блоком МРХ в очередь DIRQ.

Функция распределения памяти возложена на загрузчик. Как уже отмечалось выше, запросы выбираются в порядке снижения заказов на объем памяти. Следовательно, в случае, если память не может быть выделена для данного запроса, его необходимо отложить до окончания одной из решаемых задач. По окончании этой задачи память может оказаться достаточной для отложенного запроса. Следовательно, на блок окончания задачи, входящий в состав МРХ, возлагается функция распознавания ситуации, когда дальнейшая работа загрузчика прекращена по этой причине. Если установлено, что это так, то этот же блок выводит загрузчик из активного состояния.

Как уже отмечалось, МРХ должен реагировать на требование выдачи данных на печать. В этом случае функция МРХ заключается в том, чтобы определить, находится ли распределитель в активном состоянии, и если нет, то перевести его в это состояние. МРХ должен также поместить запрос на вывод от рабочей программы в очередь распределителя.

Выдача данных для такой рабочей программы не начнется до тех пор, пока для этой программы не будет работать блок окончания задачи, что указывает на то, что по данной программе все вычисления закончены. Когда это происходит, блок окончания задачи должен активизировать один из блоков выдачи сообщений (если есть неработающий в данный момент блок), а затем поместить запрос на окончание рабочей программы в очередь диспетчера.

### **Блок-схема монитора**

Несколько подробнее опишем некоторые элементы ОС. Метод, используемый нами для анализа запросов, введен в главе 5. Таким образом, с каждой программой, входящей в состав операционной системы и выполняющей определенную независимую функцию, связан свой собственный запрос и поле запроса. Исходя из этого,



можно сказать, что каждая из блок-схем показывает обрабатываемый запрос, как запрос, связанный с определенной программой операционной системы. На этих схемах будут показаны запросы в различных логических ситуациях.

Программа ОС называется *независимой*, если ей соответствует свой собственный запрос. Программа называется *асинхронной*, если этот запрос обрабатывается также в мультипрограммном режиме. Программы, способные выполняться при таких условиях, могут быть описаны независимо одна от другой с указанием связей между ними, идущих через соответствующие очереди, и описанием данных, содержащихся в полях запросов. Короче говоря, такие программы описываются и работают независимо одна от другой.

## Приемник

Эта программа имеет свой собственный запрос и поле запроса. Обозначим ячейку оперативной памяти, отведенную под поле запроса приемника, через R. Она же будет и идентификатором поля запроса, а, следовательно, будет появляться в качестве символа для слова запроса приемника. Кроме того, есть еще одна фиксированная ячейка памяти, чей символический адрес будет R1. В этой ячейке содержится слово запроса приемника, если эта программа находится в неактивном состоянии. Эти две ячейки памяти показаны на рис. 18.

В общем виде последовательность действий при вводе задания в систему состоит в следующем. Данные, содержащие описание работы и параметры, загружаются в устройство ввода 1, после чего нажимается кнопка запроса на ввод, передающая сигнал прерывания в операционную систему. Это прерывание принимается MPX и интерпретируется им как сигнал о наличии колоды перфокарт в устройстве ввода. Задание на работу от пользователя ОС есть описание действий, которые пользователь ждет от операционной системы. С другой стороны, это сигнал для операционной системы о присутствии пользователя в ней. После задания на работу следует переход к его обработке и загрузке программ, к выполнению этих программ и выводу данных по ним.

Следовательно, по получении такого прерывания МРХ генерирует запрос на прием этого задания.

Генерация запроса по данному заданию включает в себя помещение поля запроса в определенное слово памяти и использование адреса этого слова в качестве запроса. Из этой интерпретации по новому заданию МРХ создает слово запроса, предназначенное для приемника. Это производится путем помещения вновь созданного слова запроса в очередь приемника RCVQ. Положение нового слова в очереди зависит от времени его поступления.

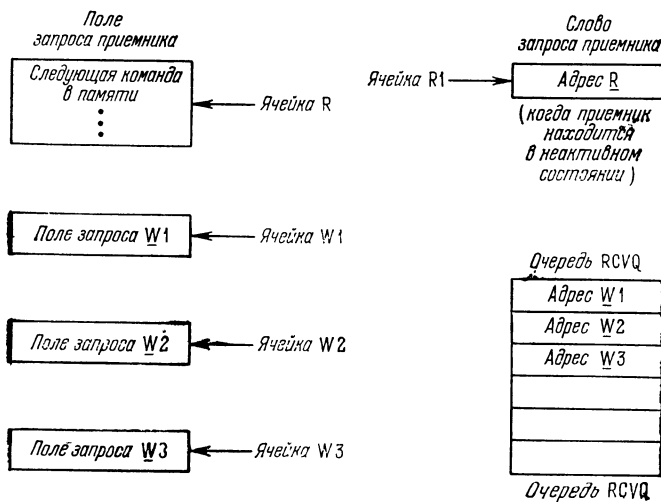


Рис. 18. Запросы и очередь RCVQ.

Процесс, описанный выше, является общим для всех поступающих от пользователей заданий, помещаемых в устройство ввода, о наличии которых сообщается прерыванием, направляемым в МРХ. Действия, описанные выше, являются независимыми от асинхронной работы самого приемника. Таким образом, работа программы-приемника по обработке очередного запроса из очереди RCVQ может прерываться из-за поступления нового задания в устройство ввода. Приемник в этот момент никак не реагирует на появление нового запроса. Так как его работа полностью определяется наличием

слова запроса в очереди RCVQ, то асинхронное появление нового запроса в очереди не влияет на его работу.

На рис. 18 показана ситуация, в которой создано 3 запроса в очереди. Поля запросов помещаются в символических ячейках W1, W2, W3, а соответствующие им слова запросов помещены в очередь RCVQ.

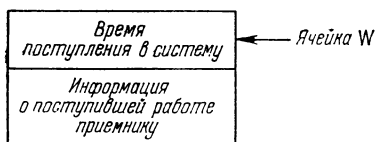


Рис. 19. Поле запроса для нового задания.

В этой ситуации активизация приемника производится монитором, который помещает слово запроса приемника в очередь имеющихся работ, делая его, таким образом, доступным для обработки. Начиная ра-

ботать, приемник выбирает верхнее слово запроса из RCVQ, читает соответствующие ему данные, выполняет действия, которые будут описаны ниже и, наконец, опрашивает очередь RCVQ. Если она не пуста, программа повторяет работу для второго запроса из очереди. Этот цикл повторяется до тех пор, пока очередь RCVQ не станет пустой, как уже отмечалось. В ходе выполнения этой работы очередь RCVQ может пополняться монитором, благодаря прерываниям, поступающим с устройства ввода.

Структура поля запроса для вновь поступившего задания до его обработки приемником показана на рис. 19.

Как видно из рисунка, MPX поместил данные о времени поступления задания с устройства ввода в поле запроса. Остальное место в поле запроса пока пусто. Оно будет использовано приемником для размещения соответствующей информации о задании.

Блок-схема программы-приемника показана на рис. 20. Приемник активизируется монитором, когда его слово запроса помещается в очередь задач, ожидающих решения. Если приемник выбран в решение, его слово запроса помещается из AQ в COT. Обозначим через R запрос приемника. Так этот запрос будет изображаться на всех блок-схемах.

Работа приемника начинается в точке 1. Так как приемник является той программой, которая работает

в текущий момент, то значение COT (текущий обрабатываемый запрос) равно содержимому  $R$ .

Для приемника в этой точке не существует никаких других запросов. Первым действием приемника является выбор запроса из очереди приемника. Он станет

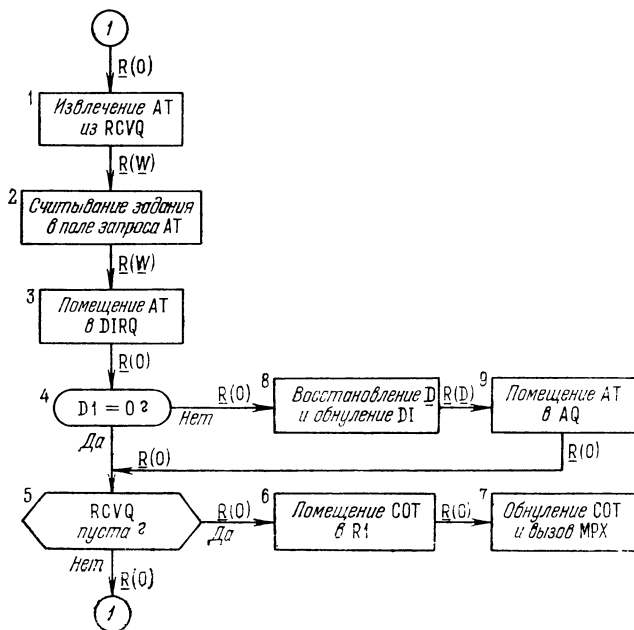


Рис. 20. Блок-схема программы-приемника.

словом запроса принимаемого задания. По окончании этой операции запрос станет таким, как указано на выходе блока 1. Мы видим, что  $R$  является запросом, обрабатываемым в настоящий момент, и что  $W$  — задание, обрабатываемое приемником.

Следующим действием приемника (блок 2) является чтение информации о задании пользователя. Эта информация помещается в ячейки, отведенные для этого в поле запроса. Слово запроса  $W$  является по существу адресом этого поля. Таким образом, при работе блока 2 приемник имеет всю необходимую информацию для выделения ячеек под ввод с перфокарт. Функцией блока 2 является выполнение операций по вводу. Выполнение

этих операций производится модулем ввода-вывода. В это время МРХ рассматривает программу-приемник как любую другую задачу, работающую под его управлением. Следовательно, запрос приемника  $R$  поступает через МРХ в очередь, связанную с каналом, соединенным с устройством ввода с перфокарт. В ходе этого запрос получает доступ к каналу и на время чтения карт находится в списке слов, соответствующем выбранному каналу.

В это время запрос приемника не может оказаться в очереди АQ, и поэтому программа-приемник работать не может. Когда работа по вводу закончена, слово запроса  $R$  появится в АQ и может быть выбрано в качестве СОТ. Это вызовет возврат к блоку 2, с которого программа-приемник и возобновит свою работу.

Непосредственно перед передачей  $R$  в МРХ для выполнения операций по вводу карт слово запроса  $W$ , находящееся в ячейке АТ, упрятывается. Это слово при выходе из МРХ вновь восстанавливается, раз в качестве обрабатываемого выбран запрос  $R$ . Таким образом, при входе в блок 3 состояние таково, что вновь созданный запрос  $W$  опять стал обрабатываемым запросом. Информация о задании пользователя находится в это время в соответствующем  $W$  поле запроса. Следующим действием по этому запросу является сообщение директору о присутствии запроса. Это достигается помещением текущего запроса в очередь директора DIRQ (блок 3). Директор обрабатывает каждый запрос из очереди DIRQ и, следовательно, он получит запрос, помещенный в его очередь. Раз запрос из АТ помещен в очередь DIRQ, то он больше не является запросом в состоянии готовности для  $R$ , как это и указано на выходе из блока 3.

В блоке 4 приемник определяет, находится ли директор в активном состоянии. Если нет, то запрос директора  $D$  должен находиться в ячейке D1. Если директор в активном состоянии, то, следовательно, директор является одной из программ, находящихся в ведении МРХ. В этом случае ячейка D1 должна быть пустой. Предположим, что мы столкнулись с последним случаем и из блока 4 попали непосредственно в блок 5 с АТ, равным нулю для запроса  $R$ . В блоке 5 определяется, пуста ли очередь приемника. Если нет, то

она содержит по крайней мере еще один запрос от пользователя, который должен обрабатываться программой-приемником. Отметим, что на выходе из блока 5 состояние ЦП описывается как  $R(0)$ , что является условием перехода к точке 1 для возобновления работы приемника.

Вернемся снова к блоку 5. Если очередь приемника пуста, то дальнейшие работы по запросу  $R$  проводятся с целью выведения его из активного состояния. В блоке 6 приемник помещает COT в ячейку  $R1$ . Это означает удаление запроса  $R$  из обработки. Наличие запроса в  $R1$  означает для MPX то, что приемник находится в неактивном состоянии. Как на входе, так и на выходе из блока 6 нет текущего запроса. Последними действиями приемника являются обнуление ячейки текущего запроса и вызов MPX через ячейку MPX, назначение которой будет рассмотрено позже.

В случае, если приемник определил, что директор находится в неактивном состоянии после того, как новый запрос был помещен в очередь DIRQ, ячейка D1 должна содержать запрос директора и, следовательно, должна быть отлична от нуля. Тогда приемник, оставляя ячейку AT пустой, переходит к блоку 8. Здесь приемник выбирает запрос директора из ячейки D1 и затем обнуляет ее, показывая тем самым, что директор занят. Теперь запрос директора  $D$  является запросом в состоянии готовности, обработка которого выполняется под управлением запроса  $R$ . Следовательно, запрос  $D$  появится в поле AT. Блок 9 завершает перевод директора в активное состояние, помещая его запрос в очередь задач, ожидающих решения, откуда он может быть выбран монитором в работу. При перемещении запроса из поля AT в очередь AQ AT обнуляется, как указано на выходе блока 9.

Переведя директор в активное состояние, приемник переходит к блоку 5, работа которого уже описывалась.

Важно отметить, что основными логическими моментами в работе приемника являются обработка вновь поступившего запроса пользователя и запроса директора. Эти слова запросов позволяют косвенно обратиться к соответствующим полям запросов. Таким образом, приемник имеет всю информацию, необходимую для его работы. Более важным, однако, является то, что

программа-приемник работает асинхронно со всеми другими программами ввода-вывода или программами пользователя. Более того, они все работают асинхронно с монитором, который асинхронно заполняет поля запросов и слова запросов для вновь полученных заданий пользователей и помещает запросы в очередь RCVQ.

Это происходит благодаря работе программы-приемника. Помещение слов запросов в очередь RCVQ происходит без какой-либо синхронизации.

## Директор

Как и в случае приемника, слова запросов директора появляются в очереди DIRQ асинхронно с работой как самого директора, так и остальных программ ОС. Эти запросы, которые появляются в очереди DIRQ, возникают в результате работы приемника. Если директор находится в активном состоянии, его слово запроса помещается приемником в очередь задач, ожидающих решения. Оно может быть затем выбрано монитором в качестве COT. Если это происходит, то работа программы-директора начинается с точки 1 (рис. 21, стр. 127). В этом случае запрос директора является текущим обрабатываемым запросом в поле COT, а поле AT пусто.

Директор находится в активном состоянии, так как в его очереди DIRQ имеется по крайней мере один запрос, представляющий задание пользователя. В блоке 1 выбирается первое слово запроса из DIRQ и помещается в поле AT. Запрос, ставший, таким образом, доступным для обработки директором, обозначен буквой *T* на выходе из блока 1.

Задачей директора на данном этапе является распознавание типа работы (ввод, компиляция и т. д.). Для этих целей служит блок 2. В блоке 3 опрашивается содержимое запроса работы. Предположим, что это не работа по чтению перфокарт. Тогда запрос наряду с запросом директора передается в блок 4 для определения того, не является ли это работой по компиляции. Предположим опять, что это не так. Тогда программа переходит в точку 6.

К этому моменту уже известно, что обрабатываемый запрос не требует ни работы по вводу, ни компиляции. В блоке 5 (рис. 21, стр. 128) определяется, свободны ли

устройства, необходимые для этого запроса (они определены при интерпретации задания в блоке 2). Если устройства свободны, то управление передается в блок 6, где АТ (запрос  $T$ , обрабатываемый директором) помещается в очередь загрузчика. Таким образом, по окончании работы блока 6 директор уже определил, что

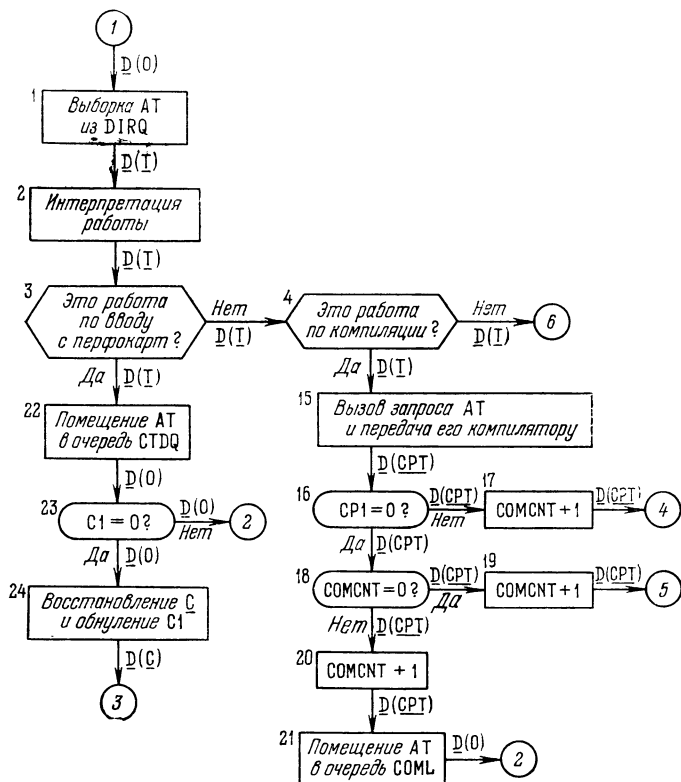


Рис. 21. Директор.

поступивший запрос не требует ни работы по вводу с карт, ни операций по сборке, и что устройства, необходимые для выполнения работ по этому запросу, свободны. Следовательно, в блоке 6 запрос на выполнение помещается в очередь загрузчика LDRQ для загрузки необходимых программ и последующего выполнения.



Выполнив это, директор теряет обрабатываемый запрос, что и указано на выходе из блока. Так как директор поместил слово запроса в очередь загрузчика, он должен определить, находится ли загрузчик в активном состоянии. Это выполняется в блоке 7.

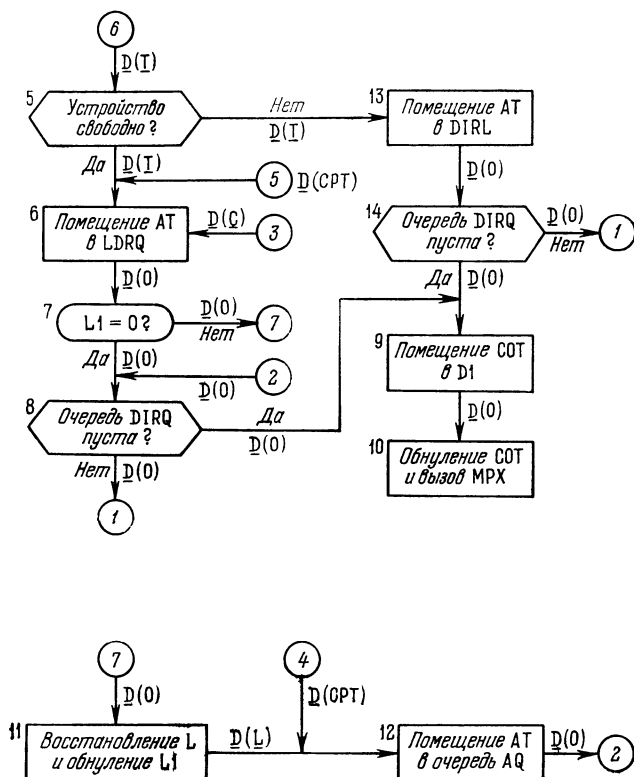


Рис. 21. Директор (продолжение).

Если загрузчик не в активном состоянии, то его слово запроса должно находиться в ячейке L1. Следовательно, если содержимое L1 равно нулю, то загрузчик находится в активном состоянии и работает. В этом случае управление передается в блок 8, так как никаких действий по активизации загрузчика выполнять не нужно.

В блоке 8 директор проверяет, есть ли еще запросы в его очереди. Если да, то это означает, что еще одна работа требует внимания со стороны директора. В этом случае управление передается в точку 1, и работа директора повторяется. Заметим, что в этот момент поле АТ пусто.

Если в очереди директора нет работ, требующих интерпретации, тогда управление передается в блок 9. От директора не требуется больше выполнение его функций. В этом блоке директор переводит себя в неактивное состояние, помещая свое слово запроса в ячейку D1. В блоке 10 содержимое COT обнуляется и вызывается MPX.

Вернемся к блоку 7. Предположим, что поместив запрос в очередь загрузчика, директор установил, что загрузчик находится в неактивном состоянии. Это устанавливается путем проверки содержимого ячейки L1. В нашем случае в L1 должно содержаться слово запрос загрузчика. В этой ситуации управление передается в точку 7, являющуюся входом в блок 11. В этом блоке запрос загрузчика  $L$  восстанавливается, а ячейка L1 обнуляется, указывая на перевод загрузчика в активное состояние. Заметим, что обрабатываемым запросом является теперь запрос загрузчика. Далее в блоке 12 директор помещает этот запрос в очередь AQ. На этом и заканчивается активизация загрузчика.

На выходе из блока 12 указано, что больше нет обрабатываемого запроса. При переходе в точку 2 мы входим в блок 8, где определяется, остаются ли директору в активном состоянии или нет.

При выполнении действий по обработке запроса на работу, отличную от ввода с карт или сборки, может оказаться, что устройства, необходимые для данной работы, заняты. В этом случае из блока 5 мы попадаем в блок 13. Здесь запрос  $T$  (запрос в состоянии готовности) помещается в список директора D1RL, где он будет ожидать окончания текущих работ, в результате чего могут освободиться необходимые устройства. На этом действия директора по обработке данного запроса приостанавливаются до окончания выполнения текущих программ.

Так как в блоке 13 директор поместил готовый к обработке запрос в свой список, то не стало обрабаты-

мого запроса, что и указано на выходе из блока 13. При этих условиях управление передается в блок 14, где проверяется, пуста ли очередь DIRQ. Если очередь не пуста, то это означает наличие работы, требующей внимания директора. В этом случае управление передается в точку 1.

Если очередь директора пуста, управление передается в блок 9, где слово запроса директора помещается в D1, обнуляется ячейка COT (блок 10) и управление переходит к MPX.

До сих пор мы рассматривали случай, когда запрос *T* представлял запрос на работу, отличную от компиляции или ввода с перфокарт. Теперь мы рассмотрим случай (блок 4), когда запрос *T* является запросом на сборку программы, написанной в кодах. В этом случае запрос *D* вместе с запросом *T*, представляющим запрос на сборку, передается в блок 15.

Как уже отмечалось ранее, компилятор является, во-первых, повторно входимым и, во-вторых, трактуется как рабочая программа. Здесь необходимо связать запрос на компиляцию с самим компилятором. Таким образом, запрос *T* объявляется запросом на обработку компилятором. Для этого необходимо только в поле запроса указать имя программы-компилятора (вопрос о постоянном присутствии компилятора в оперативной памяти будет рассмотрен ниже). Следовательно, действия компилятора производятся на основе запроса, который использует компилятор в качестве обслуживающей программы.

Поэтому на выходе из блока 15 в качестве обрабатываемого (текущего) запроса указано слово запроса компилятора. На блок-схеме это слово обозначено через *CPT*.

Переключатель CP1 используется для выяснения того, находится компилятор в оперативной памяти или нет. Если компилятор в памяти, то CP1 отлично от нуля и равно нулю в случае отсутствия компилятора. Предположим теперь, что компилятор находится в оперативной памяти ( $CP1 \neq 0$ ). Управление передается в блок 17, где счетчик COMCNT увеличивается на единицу. Этот счетчик указывает число одновременно обрабатываемых компилятором заданий и служит для определения того момента, когда компилятор можно убрать из оператив-

ной памяти. С этим увеличенным на единицу значением счетчика запрос директора передается в точку 4, являющуюся входом в блок 12. Так как компилятор находится в оперативной памяти ( $CP1 \neq 0$ ) и так как он обладает свойством повторной входимости, он может сразу начать работать. Таким образом текущий запрос помещается в очередь AQ (блок 12), откуда он может быть выбран монитором в качестве COT.

При выходе из блока 12 ячейка AT запроса пуста, так как этот запрос перенесен в очередь AQ задач, ожидающих решения. Управление передается в точку 2, являющуюся входом в блок 8.

Вернемся теперь к блоку 16. Предположим, что ключ  $CP1$  установлен в 0. В этом случае компилятор отсутствует в оперативной памяти и выполняется одно из двух условий. Первое: компилятор может быть считан, и, следовательно, действующий запрос будет использован для формирования вызова загрузчика для перезагрузки компилятора в память. Второе: для этой цели может быть использован предыдущий запрос, чтобы в настоящее время компилятор находился в процессе загрузки. В этом случае директор запомнит запросы, ожидающие работы компилятора до момента окончания процесса загрузки этой программы.

В блоке 18 директор проверяет, загружается ли копия компилятора. Это достигается проверкой значения счетчика  $COMCNT$ . Так как компилятора нет в памяти ( $CP1 = 0$ ) и если  $COMCNT$  равен нулю, то ни один из предыдущих запросов не вызвал загрузки компилятора. Тогда управление с  $CPT$  (запрос компилятора) в качестве AT переходит в блок 19, где счетчик числа программ, ожидающих обслуживания компилятором, увеличивается на единицу, после чего управление передается в точку 5.

Точка 5 является входом в блок 6, в котором директор помещает имеющийся запрос в очередь загрузчика и таким образом начинает загрузку программы (в данном случае компилятора) в оперативную память. Последующие действия (блок 7, 8 и т. д.) уже описывались ранее и заключаются в проверке занятости загрузчика и выяснении необходимости дальнейшей работы директора для запросов из очереди  $DIRQ$ .

Вернемся к блоку 18. Предположим, что предыдущий запрос обнаружил отсутствие компилятора в оперативной памяти и произвел поэтому описанные выше действия по его загрузке. К настоящему моменту для текущего запроса эти действия уже закончились. В этом случае, несмотря на то, что  $CP1=0$ , значение  $COMCNT$  отлично от нуля, так как предыдущий запрос увеличил его значение на единицу. Тогда управление передается в блок 20, где счетчик запросов, ожидающих работы компилятора, увеличивается, и происходит переход в блок 21, где этот запрос помещается в список  $COML$ . Этот список  $COML$  используется в качестве временной памяти для запросов, ожидающих загрузки компилятора. По завершении этой операции загрузчик помещает все слова запросов из списка  $COML$  в очередь задач, ожидающих решения, откуда они впоследствии выбираются компилятором в решение. Раз готовый к обработке запрос на компиляцию помещен в список  $COML$ , управление передается в точку 2, и директор выполняет действия по определению необходимости продолжения работ с запросами из очереди  $DIRQ$ . Теперь мы рассмотрим случай, когда запрос на работу  $T$  представляет собой запрос по вводу с перфокарт. В этом случае  $T$  из блока 3 управление передается в блок 22. Здесь этот запрос  $T$  помещается в очередь на ввод с перфокарт  $CTDQ$ . Как раз из этой очереди запросов и выбирает программа ввода с перфокарт очередной запрос. Следует отметить, что эта программа может работать только последовательно по той причине, что только одно устройство ввода-2 приспособлено для ввода с перфокарт.

Так как запрос  $T$  помещен в очередь  $CTDQ$ , у директора больше нет запроса, что и указано на выходе из блока 22. В блоке 23 директор проверяет наличие в оперативной памяти программы ввода с перфокарт и готовность ее к работе. Это выполняется при помощи опроса ячейки  $C1$ . Программа ввода с перфокарт — не повторно входимая и связана со своим собственным запросом, запоминаемым в ячейке  $C1$  в случае нахождения этой программы в неактивном состоянии. Следовательно, если директор нашел, что ячейка  $C1$  не пуста, управление передается в блок 24, где слово запроса этой программы ( $C$ ) извлекается из ячейки  $C1$ , и эта

ячейка обнуляется. Этот запрос  $C$  становится готовым к обработке запросом, и управление передается в точку 3, являющуюся входом в блок 6. Директор помещает запрос  $C$  в очередь загрузчика и тем самым начинает загрузку программы ввода с перфокарт.

В случае, если программа ввода с перфокарт уже работает и, следовательно, загружена в память, управление из блока 23 передается в точку 2, после чего директором производится проверка необходимости его дальнейшей работы с запросами из очереди DIRQ. Кроме этого никакие другие действия не выполняются, так как программа ввода с перфокарт уже находится в памяти и работает, и запрос на ввод ( $T$ ) уже помещен в очередь этой программы CTDQ.

### Загрузчик

Так же как приемник и директор, загрузчик работает как независимая и асинхронная программа, входящая в состав ОС. Взаимодействие загрузчика с другими элементами ОС происходит через очередь загрузчика LDRQ.

Загрузчик является составляющей ОС, которая производит распределение памяти. Для того чтобы установить критерий для выбора следующей программы, подлежащей загрузке, предположим, что компилятор значительно больше тех программ, которые обрабатываются системой. Так как частота компиляции относительно высока в нашей предполагаемой системе, желательно иметь такой критерий для распределения памяти, чтобы он гарантировал загрузку компилятора. Для того чтобы иметь преимущество при загрузке, программа-компилятор должна быть достаточно большой, потому что загрузчик при выборе запроса из очереди LDRQ отдает предпочтение самой большой (в смысле занимаемой памяти) задаче. Более того, специально предполагается, что распределение памяти для этого самого большого запроса в очереди должно быть выполнено до загрузки всех остальных программ. Это означает, что если для самого большого запроса памяти сейчас невозможно провести распределение, то дальнейшее распределение памяти, а, следовательно, и загрузка приостанавливаются до появления возможности

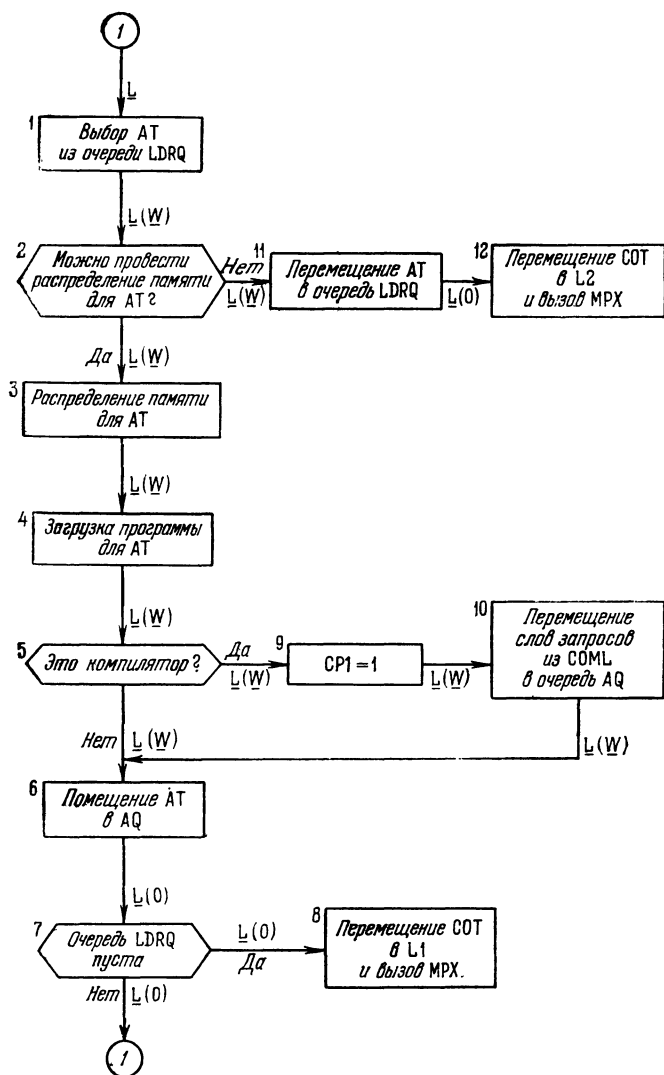


Рис. 22. Загрузчик.

удовлетворения этого запроса. Это происходит даже в том случае, когда для других запросов в очереди LDRQ распределение памяти можно было бы произвести. Установив таким образом критерий для распределения памяти, мы переходим к описанию блок-схемы и работы загрузчика (см. рис. 22).

При активизации загрузчика монитором запрос загрузчика  $L$  помещается в очередь задач, ожидающих решения, откуда он может быть выбран в качестве кандидата на решение. Когда загрузчик выбирается в решение, управление передается в точку 1.

Загрузчик начинает работу с выбора слова запроса из очереди LDRQ. Как уже отмечалось выше, это производится на основе принципа максимального запроса памяти. Выбранный таким образом запрос становится готовым к обработке запросом. Его слово запроса помещается в поле AT. Этот запрос обозначен буквой  $W$ . На выходе из блока 1 указано, что запрос загрузчика  $L$  является COT.

В блоке 2 вызывается собственно программа-распределитель, которая сопоставляет запрос на память в  $W$  с имеющейся в данный момент свободной памятью. Предположим, что распределение памяти провести можно. Тогда оно и производится в блоке 3. В блоке 4 производится загрузка программы. Фактически это означает выполнение загрузчиком операций по перемещению загружаемой программы из библиотеки на диске в оперативную память. Эти операции выполняются в мультипрограммном режиме. Таким образом, запрос  $L$  передается в MPX, запрос ввода-вывода трактуется как любой другой запрос.

После того, как программа загружена, запрос  $W$  остается в качестве готового к обработке и осуществляется переход в блок 5. Здесь происходит определение того, является ли загруженная программа компилятором. Если это не компилятор, управление передается в блок 6, где запрос  $W$  помещается в очередь задач, ожидающих решения, так как программа для  $W$  находится уже в памяти и готова к работе.

На выходе из блока 6 указано, что больше нет запроса, обрабатываемого загрузчиком, так как он находится уже в очереди AQ. Затем программа переходит в блок 7, где определяется, пуста ли очередь LDRQ.



Если очередь не пуста, осуществляется переход в точку 1 и работа загрузчика повторяется. Если же очередь пуста, то загрузчик больше не нужен и в блоке 8 его запрос  $L$  (размещенный в ячейке COT) помещается в ячейку L1, сообщая таким образом директору, что загрузчик находится в неактивном состоянии. Последним действием загрузчика является вызов MPX, который и заканчивает работу загрузчика.

Вернемся снова к блоку 5. Предположим, что загрузчик определил, что программа, для которой была только что выполнена загрузка, является компилятором. Тогда осуществляется переход в блок 9, где переключатель CP1 устанавливается в 1, сообщая тем самым директору, что компилятор находится в памяти. Следующим действием, выполняемым в блоке 10, является перемещение всех слов запросов из списка COML в очередь AQ. Эти слова запросов представляют задания на компиляцию. Каждое такое слово запроса последовательно попадает на место запроса  $W$ , который только что вызвал загрузку компилятора. Выполнив это, загрузчик переходит к блоку 6, работа которого уже описывалась.

До этого была описана ситуация, когда для запроса  $W$  было возможно произвести распределение памяти. Вернемся к блоку 2 и предположим, что при обработке запроса  $W$  с максимальным требованием памяти в очереди LDRQ было обнаружено отсутствие необходимой памяти. В этом случае происходит передача управления в блок 11. Запрос  $W$  все еще остается в AT. В блоке 11 запрос  $W$  помещается снова в очередь LDRQ, откуда он впоследствии снова выберется для обработки. Отметим, что при выходе из блока 11 поле AT пусто.

В блоке 12 загрузчик помещает содержимое COT, т. е. свой собственный запрос  $L$ , в ячейку L2 и вызывает MPX. Тем самым MPX получает сообщение о том, что загрузчик находится в активном состоянии и ожидает окончания программы. При очередном окончании программы освободится память, и загрузчик вновь попытается распределить память для запроса, которому не хватало памяти до этого. Следует подчеркнуть, что слово запроса находится в ячейке L2, а не в ячейке L1, а это означает, что загрузчик находится в активном состоянии.

Надо заметить, что к моменту возвращения запроса  $W$  в очередь загрузчика там может оказаться запрос  $W1$ , требующий больше памяти, чем  $W$ , и, следовательно, в соответствии с критерием выбора задач для загрузки, обладающий перед  $W$  приоритетом с точки зрения загрузчика.

### Блок окончания задач

Каждая рабочая программа, обрабатываемая в системе, передается монитору для окончания. Подпрограмма MPX, выполняющая функции по окончании задачи, называется блоком окончания задач. Этот блок производит все необходимые действия по перераспределению устройств и памяти, предназначенных для окончания запроса, а также начинает работу по выводу информации, полученной в результате работы программы.

Для этого используется два печатающих устройства, выбор которых производит MPX при выдаче информации по данной работе. Пакет ввода-вывода совместно с распределителем передает эту информацию в буфер печати, находящийся во внешней памяти с произвольным доступом. Выдача этой информации начнется по окончании рабочей программы.

Так как блок окончания задач несет ответственность за перераспределение устройств, он должен определить, есть ли слова запросов для рабочих программ в списке DIRL, где они ожидают доступа к устройствам до того, как будут обработаны директором. Следовательно, блок окончания задач должен определить, что освобождение того или иного устройства дает возможность директору обработать рабочий запрос. В этом случае блок окончания задач должен ~~поместить~~ все такие запросы из списка директора DIRL в очередь директора DIRQ.

Последней функцией блока окончания задач является определение случая, когда дальнейшая работа загрузчика была приостановлена из-за недостатка памяти.

Так как работа по данному запросу заканчивается и память освобождается, то эту проверку лучше всего возложить на блок окончания задач.

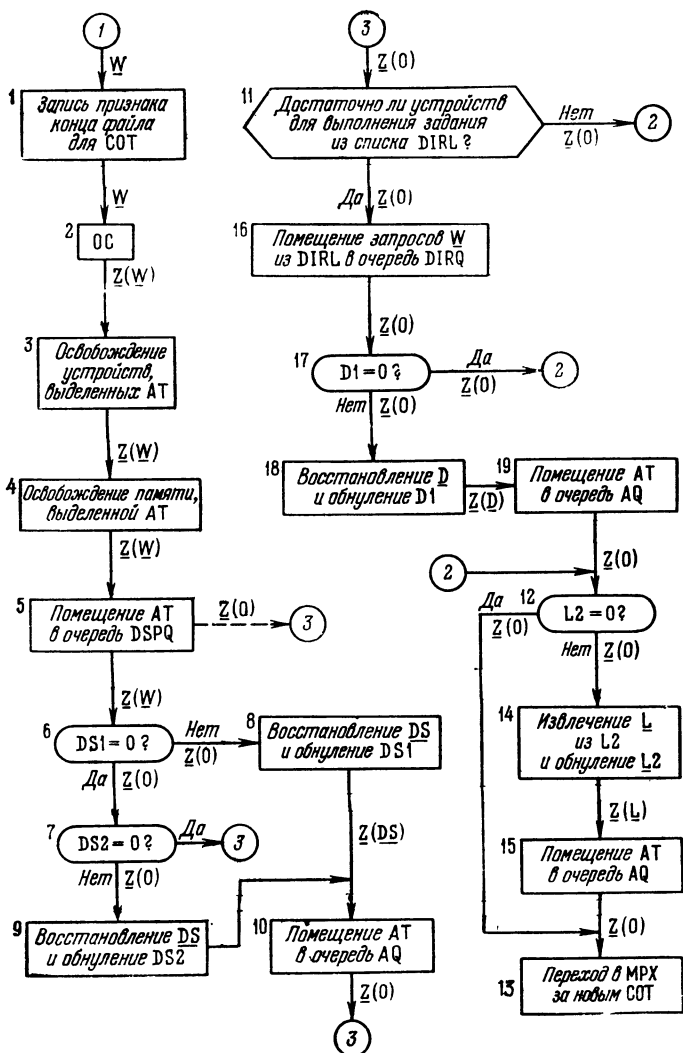


Рис. 23. Блок окончания задач.

Работа блока окончания задач изображена на блок-схеме (рис. 23). Запрос  $W$ , находящийся в стадии окончания, передается в МРХ и появляется в точке 1. Первое, что делает сам запрос (блок 1), это вызывает запись признака конца файла вывода по данному запросу. Это, конечно, предполагает, что информация для вывода была заготовлена запросом  $W$  во время его выполнения и, следовательно, блок 1 включает в себя и проверку этого. Следует подчеркнуть, что эта проверка может быть выполнена просто опросом соответствующего разряда в поле запроса  $W$ . Этот разряд устанавливается в блоке 1 программой-распределителем, вызываемой запросом  $W$  при возникновении необходимости в передаче данных в буфер вывода.

Работа блока 1 производится при воздействии рабочего запроса  $W$ . При завершении этой работы управление передается ОС (в блок 2). Как уже упоминалось в главе 5, это делает запрос текущим. Слово запроса помещается при этом в ячейку COT текущего обрабатываемого запроса, и запрещаются прерывания.

На выходе из блока 2 указано, что  $Z$  является текущим обрабатываемым запросом, а запрос  $W$ , по которому ведутся работы по окончанию, обратился к запросу  $Z$ . Следовательно, при входе в блок 3 устройства, выделенные запросу  $W$ , освободились. В блоке 4 освобождается память, выделенная запросу  $W$ , а в блоке 5 слово запроса помещается в очередь DSPQ для обработки диспетчером. Естественно, что если в блоке 5 будет определено, что по этому запросу нет выдачи на печать, то управление передается непосредственно в точку 3.

Предположим, что этот запрос имеет вывод на печать. Тогда его слово запроса помещается в DSPQ, и, таким образом, состояние системы определяется выражением  $Z(0)$ . Так как при входе в блок 6 имеется по крайней мере один запрос в очереди DSPQ, то блок окончания задач должен определить, занят ли диспетчер. Выражение *диспетчер занят* означает, что каждый из двух запросов диспетчера (по одному на каждое устройство печати) обрабатывается в настоящий момент в мультипрограммном режиме. Следовательно, если диспетчер занят, блок окончания задач не найдет запросов ни в ячейке DS1, ни в ячейке DS2 (там находятся

запросы диспетчера, если он не занят). В блоке 6 проверяется ячейка DS1, а в блоке 7 — ячейка DS2 на наличие в них запросов диспетчера. Если запросов нет, то состояние системы описывается выражением  $Z(0)$  и управление передается в точку 3, чтобы затем перейти в блок 11, так как диспетчер находится в активном состоянии и оканчивающийся запрос уже находится в очереди DSPQ.

Если какая-либо из ячеек DS1 и DS2 не равна нулю, то по крайней мере одно из слов запроса диспетчера свободно и управление передается либо в блок 8, либо в блок 9, в зависимости от того, какое слово свободно. Запрос диспетчера для одного из двух устройств печати восстанавливается так, что при входе в блок 10 состояние системы описывается выражением  $Z(DS)$ .

Блок 10 помещает выбранный запрос диспетчера (теперь это AT) в очередь задач AQ, ожидающих решения, откуда он может быть выбран монитором для выполнения программы диспетчера. При выходе из блока 10 уже нет больше запроса в состоянии готовности. Из блока 10 управление передается в блок 11 (через точку 3).

Теперь память и устройства, выделенные запросу W, уже возвращены системе, и запрос рабочей программы помещен в очередь диспетчера. Диспетчер уже активизирован (частично или полностью). В блоке 11 блок окончания задач выясняет, достаточно ли теперь периферийных устройств, освобожденных в блоке 3, для продолжения обработки запросов задания из списка диспетчера DURL. Предположим, что устройств даже с освобожденными в блоке 3 не хватает для продолжения обработки любого из запросов в списке DURL. В этом случае управление передается в блок 12.

В блоке 12 проверяется содержимое ячейки L2, в которой содержится запрос загрузчика L, когда работа загрузчика приостанавливается из-за невозможности произвести распределение памяти. Если эта проверка покажет, что блок окончания не должен выполнять никаких действий по отношению к загрузчику, управление передается в блок 13. Этот блок является частью монитора, в которой происходит выбор текущего обрабатываемого запроса.

С другой стороны, если окажется, что запрос  $L$  присутствует в ячейке  $L2$ , и, следовательно, его работа была приостановлена, управление передается в блок 14. Здесь слово запроса загрузчика восстанавливается из ячейки  $L2$ , а ячейка  $L2$  обнуляется. При выходе из блока 14 состояние системы описывается выражением  $Z(L)$ .

В блоке 15 запрос  $L$  помещается в очередь задач, ожидающих решения, что дает возможность загрузчику в дальнейшем начать работу. При этом загрузчик выберет из очереди  $LDRQ$  то слово запроса, для которого требуется максимальная из всех запросов память, и попытается произвести распределение памяти. Следовательно, при выходе из блока 15 ячейка  $AT$  пуста, и управление передается запросом  $Z$  (запрос операционной системы) в блок 13, назначение которого уже описывалось.

Вернемся к блоку 11. Предположим, что после освобождения устройств по окончании работы по запросу  $W$  стало возможным продолжить обработку запросов из списка  $DIRL$ . В этом случае управление передается запросом  $Z$  в блок 16, где все такие запросы из  $DIRL$  помещаются снова в очередь  $DIRQ$  для последующей обработки. После этого поле  $AT$  остается тем не менее пустым, что и указано на выходе из блока 16.

В блоке 17 производится проверка занятости директора путем опроса ячейки  $D1$ . Если запрос директора  $D$  отсутствует в ячейке  $D1$ , то значит, директор работает, и тогда управление передается в точку 2, где выполняются уже описанные действия.

Если же директор в настоящее время не работает, то ячейка  $D1$  будет отлична от нуля и управление будет передано в блок 18. В этом блоке запрос директора  $D$  будет выбран из ячейки  $D1$ , а ячейка будет обнулена, указывая на переход директора в активное состояние. При выходе из блока 18  $D$  является запросом в состоянии готовности.

В блоке 19 запрос  $D$  помещается в очередь задач, ожидающих решения. После этого у запроса операционной системы  $Z$ , выполняющего работу по окончанию задачи, больше нет работы. Управление передается в точку 2, где блок окончания задач будет исследовать некоторые требования к обслуживанию запроса загрузчика.

Вышеописанные действия по окончанию задачи выполняются для того, чтобы вернуть системе ресурсы, выделенные этой задаче. Если оканчивающееся задание подготовило данные для вывода на печать, блок окончания задач помещает слово запроса этого задания в очередь диспетчера, активизируя при этом (если необходимо) программу-диспетчер. Помимо этого, все запросы, ожидающие освобождения периферийных устройств, из списка директора DIRM перемещаются в очередь DIRQ, если освободившихся устройств достаточно для продолжения работы с этими запросами. И, наконец, блок окончания задач активизирует директор, если необходимо, и разрешает загрузчику продолжить работу.

### **Распределитель**

Задача распределителя заключается в организации выдачи по данной рабочей программе. Данные для выдачи содержатся в буфере печати. Единицей данных, передаваемых из распределителя в буфер, является блок. Каждый блок состоит из нескольких записей, размер которых ограничен длиной печатной строки устройства. Число записей в блоке может меняться от программы к программе. Это зависит от требований, предъявляемых к каждой программе. Для простоты можно предположить, что сама рабочая программа ведет счет строкам, помещаемым, на печатную страницу. Следовательно, когда рабочая программа производит выдачу последней строки на странице, она, по нашему предположению, формирует признак конца страницы. Распределитель проверяет каждую запись, полученную от рабочей программы на наличие в ней признака конца страницы. В случае обнаружения такого признака распределитель предоставляет такую запись в распоряжение диспетчера. В этот момент распределитель считает блок заполненным и помещает его в следующую область внешнего буфера (например, диска).

Понятно, что для более эффективной работы программ распределителя и диспетчера требуется некоторая организация данных в этом буфере. Существует много подходов к решению этой проблемы, каждый из

которых зависит, в основном, от рабочих характеристик самого диска. Так как разработка оптимальных способов выбора запоминающих устройств не является целью данной книги, мы можем только сказать, что организация самого буфера и блоков внутри буфера должна быть такой, что бы позволить распределителю связать следующий блок, полученный из рабочей программы, со следующей ячейкой внутри буфера. Эти ячейки блоков в буфере должны быть доступны диспетчеру в порядке их заполнения.

Программа-распределитель работает независимо и асинхронно от всех других программ, работающих в мультипрограммном режиме. Поэтому эта программа имеет свое собственное слово запроса, обозначенное на блок-схеме рис. 24 через *DT*. Когда этот запрос активизируется, он помещается в очередь задач, ожидающих решения (это действие выполняется модулем ввода-вывода), откуда он может быть выбран в качестве *COT*. Когда это происходит, управление передается в точку 1 (рис. 24).

Запросы рабочих программ, требующих внимания со стороны распределителя, находятся в очереди распределителя *DTRQ*, куда они вмещаются асинхронно во время работы модуля ввода-вывода. Первое, что делает распределитель,— это выбирает очередной запрос из очереди *DTRQ*. Он становится обрабатываемым запросом, что и указано на выходе из блока 1.

В блоке 2 распределитель включает признак вывода в поле обрабатываемого запроса. Этот признак указывает блоку окончания задач на необходимость вывода по данной рабочей программе. Эти действия не изменяют состояния системы, и поэтому оно описывается выражением, указанным на выходе из блока 2.

В блоке 3 распределитель помещает запись, подлежащую выводу по данной рабочей программе, в блок, связанный с обрабатываемым запросом. В блоке 4 распределитель проверяет запись на наличие в ней признака конца страницы.

Предположим, что в записи, подлежащей выводу, нет признака конца страницы. Тогда управление передается в блок 5, где обрабатываемый запрос помещается в очередь *AQ*. Это делает этот запрос доступным для обработки *MPX*. Текущим обрабатываемым



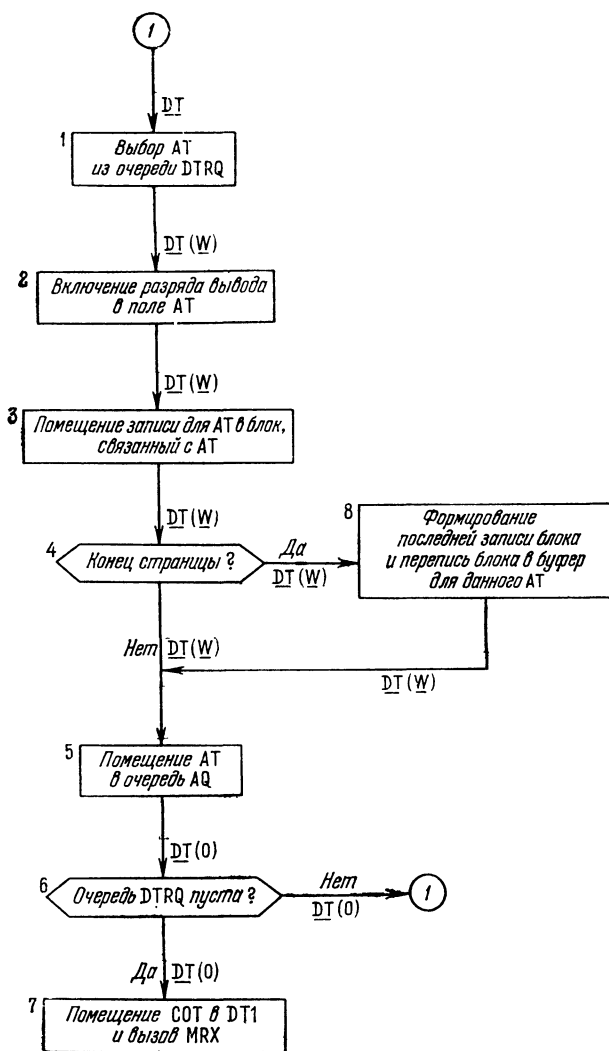


Рис. 24. Распределитель,

запросом остается *DT*. Фактически это означает, что распределитель может работать асинхронно в мультипрограммном режиме и, что наиболее важно, может обслуживать одновременно несколько рабочих программ, как это описывается ниже.

Так как распределитель поместил обрабатываемый запрос в очередь *AQ*, ячейка *AT* стала пустой; это указано на выходе из блока 5. Блоком 6 производится проверка очереди *DTRQ* на наличие в ней запросов к распределителю. Если очередь пуста, управление передается в блок 7, где распределитель помещает свое собственное слово запроса *DT* (находящееся в настоящий момент в ячейке *COT*) в ячейку *DT1* (неактивное состояние). Затем вызывает через специальную ячейку *MPX*, который и заканчивает работу распределителя. Если очередь *DTRQ* не пуста, управление передается в точку 1, и работа распределителя возобновляется.

Вернемся к блоку 4. Предположим, что запись, поступившая к распределителю, содержит признак конца страницы. В этом случае управление передается в блок 8, где распределитель формирует последнюю запись и переписывает собранный блок в буфер, связанный с данным *AT*. Эти действия не изменяют состояния системы, и поэтому при выходе из блока 8 *W* остается обрабатываемым запросом.

## Диспетчер

Диспетчер считывает блок данных из буфера печати на дисках, где каждая запись в блоке представляет собой строку устройства печати. В нашей предполагаемой системе имеется 2 таких устройства и поэтому может быть 2 запроса (2 слова и 2 поля запросов), связанных с программой-диспетчером, в предположении, что эта программа должна быть повторно входимой. Каждый из этих двух запросов связан с одним из устройств печати и, как указывалось выше, активизируется блоком окончания задач в момент окончания выполнения рабочей программы.

Во время выполнения рабочей программы, подготовившей выдачу на печать, происходит обращение к распределителю (через пакет ввода-вывода), который организует выводные данные в блоки, заканчивающиеся

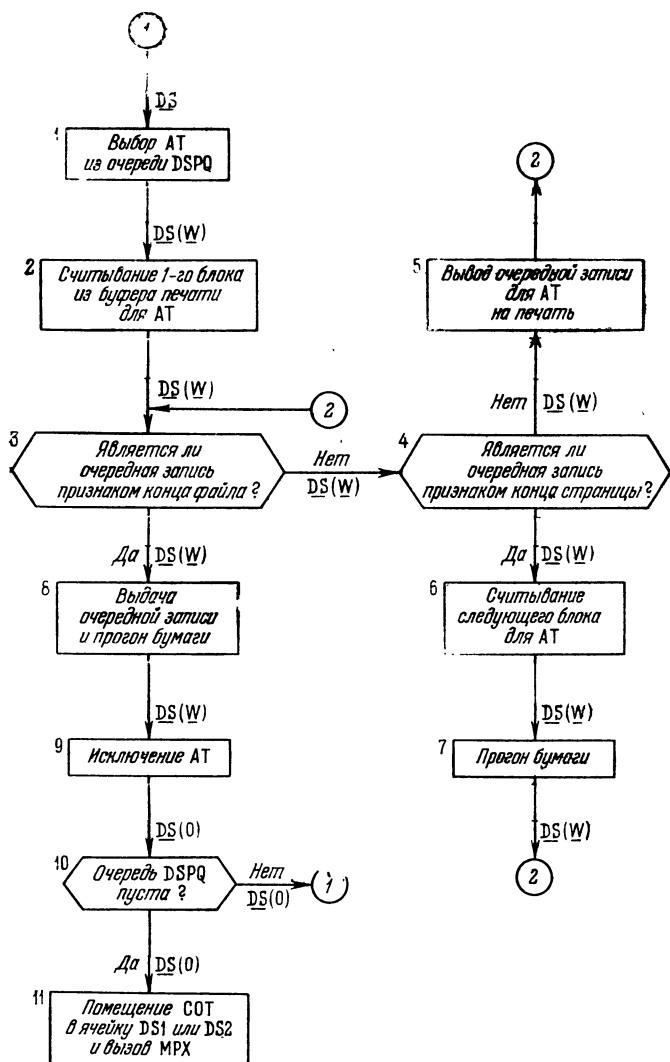


Рис. 25. Диспетчер.

признаком конца страницы. Во время окончания рабочей программы блок окончания задач передает слово запроса рабочей программы в очередь диспетчера DSPQ. Слова запросов помещаются в эту очередь асинхронно с работой двух запросов диспетчера, которые выбирают следующий запрос из DSPQ по окончании работы, как описано выше.

Диспетчер начинает свою работу в точке 1 (рис. 25), где активизируется один из запросов диспетчера. Начало работы происходит либо благодаря активизации выбранного запроса диспетчера блоком окончания задач, либо по окончании предыдущей работы диспетчера и при наличии запросов в очереди DSPQ.

В блоке 1 диспетчер выбирает запрос из очереди DSPQ и делает его готовым к обработке запросом путем помещения в ячейку АТ. В блоке 2 считывается первый блок для рабочей программы из буфера печати, и в блоке 3 начинаются операции по печати. В блоке 3 определяется, не является ли очередная запись признаком конца файла, т. е. не окончена ли уже выдача по данной рабочей программе. Эта запись извлекается, конечно, из блока буфера печати, который уже находится в памяти и обрабатывается.

В случае, когда очередная запись блока не является признаком конца файла, управление передается в блок 4, где диспетчер проверяет запись на наличие в ней признака конца страницы. Если признак конца страницы отсутствует, то эта запись выдается на устройство печати, выделенное для данного запроса, и управление передается в точку 2 для выполнения таких же действий над следующей записью в блоке.

Теперь предположим, что в блоке 4 диспетчер определил, что очередная запись является признаком конца страницы. В этом случае управление передается в блок 6, где производится считывание следующего блока из буфера печати, после чего (в блоке 7) диспетчер осуществляет прогон бумаги на устройстве печати, предназначенном для данного запроса. После выполнения этой операции управление передается в точку 2 для выполнения операций печати только что полученного из буфера блока печати.

Описанные выше операции продолжаются до тех пор, пока не встретится конец файла. Это означает

конец работы по выводу для данной программы пользователя. Конец файла генерируется программой пользователя до входа в блок окончания задач. Следовательно, в случае обнаружения блоком 3 признака конца файла управление передается в блок 8, где эта последняя запись файла (содержащая сообщение об окончании файла) выдается на печать, и производится прогон бумаги. После окончания этих действий все работы по данному запросу считаются выполненными. Поэтому в блоке 9 слово запроса (и поле запроса) для АТ могут быть уничтожены, и тем самым вся память, занимаемая этим запросом, возвращена системе. Далее диспетчер производит проверку очереди DSPQ. Если очередь DSPQ не пуста, происходит передача управления в точку 1, где происходит выбор запроса из очереди DSPQ и возобновление операций вывода.

В случае отсутствия запросов в очереди DSPQ, как и в других программах ОС, в блоке 11 диспетчер помещает свое слово запроса из ячейки COT в ячейку DS1 (или DS2), предназначенные для хранения запросов диспетчера, когда он находится в неактивном состоянии. И последней операцией, выполняемой диспетчером, является вход в MPX для завершения работы диспетчера.

## ГЛАВА 8

### РАБОТА С БУФЕРИЗАЦИЕЙ И ЗАПРОС ВВОДА-ВЫВОДА

#### Введение

В главах 4, 5 и 7 рассматривались вопросы, связанные с обработкой без буферизации программ пользователей. При этом логика работы монитора определялась независимо от характеристик файлов, используемых работающими программами. Однако, если программы обращаются к своим файлам через буфер оперативной памяти, логика работы МРХ отличается от рассмотренной выше.

Главной причиной этого является синхронизация. При работе без буферизации любое обращение рабочей программы к вводу-выводу являлось сигналом монитору для активизации ввода-вывода либо посылался запрос на выполнение таких действий к устройствам, которые выполняют эти действия. Однако при буферизации мы сталкиваемся с несколько иной ситуацией. Здесь действия рабочей программы по вводу-выводу могут повлечь, а могут и не повлечь за собой действия МРХ по вводу-выводу. Это будет зависеть от того состояния, в котором находятся буферы файлов, к которым происходит обращение. МРХ должен установить по данному запросу на ввод рабочей программы следующее: имеются ли требуемые данные в буфере, освобождают ли выбираемые данные буфер, и если да, то содержат ли другие буферы данные, связанные с этой программой. Все это предполагает, что МРХ может по запросу на ввод-вывод совершать несколько альтерна-

тивных действий. Более того, выбор одного из этих действий зависит от текущего состояния буфера, а не от самого запроса.

Поэтому первое, что рассматривается в этой главе, — это структура буферов и их возможные состояния. Затем мы перейдем к рассмотрению вопросов, касающихся операционных систем, и описанию средств управления работой вычислительной системы с буферизацией.

### Последовательные файлы

Будем считать, что основной единицей информации при обработке данных в последовательных файлах является *запись*. Запись является совокупностью последовательных ячеек памяти, которые образуют *область данного*, содержащего определенную информацию. Последовательный набор таких записей в памяти будет называться *блоком*, а число записей в блоке *коэффициентом блока*. Совокупность блоков (необязательно расположенных последовательно друг за другом), находящихся на периферийном устройстве, мы будем называть файлом. Таким образом, блок является единицей информации обмена между программой и файлом, расположенным на периферийном устройстве. Для простоты будем предполагать,

Блок 1		Блок 2	
Запись 1		Запись 7	
2		8	
3		9	
4		10	
5		11	
6		12	

что размер записей и число записей в блоке постоянны для данного файла.

Будем считать также, что буфер и файл состоят из блоков. Таким образом, если программа является однократно, двукратно, трехкратно и т. д. буферизуемой для данного файла, то она имеет область, отведенную в оперативной памяти соответственно для одного, двух, трех и т. д. блоков этого файла. Несколько блоков при многократной буферизации связаны с последовательностью упорядочения, т. е. имеют номера 1, 2, 3 и т. д. Таким образом, каждый блок в файле имеет свой порядковый номер, т. е. логически упорядочен. Это мы и называем *последовательным файлом*.

Рис. 26. Буфер на 2 блока.

Таким образом, каждый блок в файле имеет свой порядковый номер, т. е. логически упорядочен. Это мы и называем *последовательным файлом*.

Каждый блок содержит определенное количество записей. Эти записи также последовательно упорядочены так, что все записи первого блока предшествуют записям второго блока, которые в свою очередь предшествуют записям 3-го блока, и т. д. Например, при двукратной буферизации файла, имеющего коэффициент блока, равный 6, структура упорядочения записей в файле будет такой, как указано на рис. 26.

Предположим, что файл, связанный с буферами, изображенными на рисунке, является файлом ввода. Если рабочая программа исчерпает весь блок 1, и он затем будет пополняться операционной системой, то первой записью первого блока станет 13 запись файла, второй записью станет 14 и т. д.

### Буферы ввода

Буферы ввода первоначально открываются операторами «открыть файл», которые вызывают в оперативную память  $K \times N$  записей, где  $N$  — коэффициент блока, а  $K$  — число блоков в буфере. Рабочая программа обрабатывает 1-ю, 2-ю, ...,  $N - 1$ -ю запись первого блока и затем, при следующем обращении, считывает  $N$ -ю запись этого блока.

Если рабочая программа получает входную запись при помощи ОС, перемещающей очередную запись из блока в определенную область, доступную для рабочей программы, мы будем называть такой буфер *буфером ввода первого типа*. С другой стороны, если рабочая программа получает очередную запись при помощи ОС, передвигающей счетчик чтения на начало следующей записи блока, откуда программа просто считывает эту запись, мы будем называть такой буфер *буфером ввода второго типа*.

Отличия между буфером ввода 1-го типа и буфером ввода 2-го типа накладывают отпечаток на возможности компилятора и адресную структуру самой вычислительной системы. В случае буфера ввода 1-го типа очередная запись, вызываемая рабочей программой, помещается в определенные ячейки, доступные этой программе. Компилятор обеспечивает прямой доступ к отдельным данным внутри записи. Таким образом, программа обрабатывает данные, находящиеся в стандарт-



ных ячейках, и обращение к ним происходит непосредственно, в отличие от непрямой адресации. Во втором типе буфера ввода запись не перемещается, передвигается счетчик, который затем выступает в качестве базового адреса для этой записи. Этот счетчик доступен для рабочей программы, и обращение к очередной записи происходит благодаря ему, через косвенную адресацию. Это, очевидно, требует наличия в самой ВС возможности косвенной адресации и распознавания компилятором требования на этот тип адресации.

### **Буферы вывода**

Работа буфера вывода во многих отношениях сходна с работой буфера ввода. В буфере вывода 1-го типа запись, получаемая от рабочей программы, из определенной области памяти переносится на следующее место блока в буфере вывода. Это опять тот случай, когда компилятор дает возможность программе прямо адресоваться к определенной или стандартной области памяти, отведенной этой программе. Буфер вывода 2-го типа «получает» следующую запись путем прямого указания местоположения выданных программой данных. Текущая ячейка в блоке буфера устанавливается при помощи специального счетчика, работающего в ОС. Как и в случае буфера ввода 2-го типа, все обращения программ к этой новой ячейке записи выполняются при помощи косвенной адресации. Эти обращения происходят совместно с работой ЭВМ и компилятора.

### **Первый тип ввода-вывода**

Рассмотрим работу программы, у которой один из файлов имеет первый тип буфера ввода. Для того чтобы начать использование этого буфера для ввода данных, необходимо выполнить оператор «открыть файл». Это вызовет передачу необходимого числа записей из файла, содержащегося на периферийном устройстве, в оперативную память. При последующем обращении из рабочей программы на ввод из этого файла операционная система пересылает очередную запись в определенную область памяти. Так рабочая программа может рабо-

тать без задержек до тех пор, пока не исчерпается блок буфера. Однако, если считывается последняя запись из блока, операционная система должна начать работу по пополнению этого блока буфера до того, как управление перейдет к рабочей программе. Таким образом, рабочая программа может обрабатывать последнюю запись, полученную из этого блока, до тех пор, пока продолжается параллельная работа ОС по пополнению пустого блока буфера. Когда рабочая программа завершит обработку этой записи, она обратится к ОС за следующей записью, которая будет первой записью следующего блока буфера. После того как будет получена последняя запись последнего блока, возобновятся операции по пополнению этого блока, а программа может считывать следующую запись. Этой записью будет первая запись первого блока буфера.

Понятно, что во время обработки записей и проведения операций по пополнению блоков буфера может возникнуть ситуация, когда программа обращается к записи следующего блока, а этот блок еще не пополнен начатыми ранее действиями ОС. Когда это происходит, предполагается, что эта программа не может продолжать работу до тех пор, пока не станет доступным пополняемый блок буфера. При этом работа такой программы приостанавливается, а в решение выбирается следующая задача. Таким образом, если работа программы остановлена из-за отсутствия данных в одном из файлов, то этой программе не предоставляется возможности продолжения работы по другой ветви. Этот подход имеет ряд преимуществ, наиболее важным из которых является то, что управление мультипрограммным режимом осуществляется полностью операционной системой, а не прихотью программиста и его готовностью использовать привилегированные команды.

Работа буфера вывода 1-го типа во многом сходна с работой буфера ввода 1-го типа. Рабочая программа предоставляет ОС очередную выходную запись. Эта запись физически переносится из стандартной (по отношению к рабочей программе) области памяти в следующее свободное место блока буфера вывода. После переноса последней записи в блок буфера вывода операционная система начинает операции по выводу этого блока и передает управление рабочей программе. Следующая

выходная запись рабочей программы будет первой записью следующего блока буфера вывода.

Как и в случае буфера ввода, здесь возможна ситуация, когда рабочей программе необходимо поместить очередную запись для вывода в блок буфера, из которого еще не закончен вывод. В этом случае работа программы приостанавливается, и в решение выбирается следующая программа.

Для того чтобы переписать последний блок буфера из оперативной памяти в файл, рабочая программа должна выполнить оператор «закрыть файл». ОС при этом поместит признак конца файла после последней записи в этом блоке и начнет операции по переписи этого блока в файл.

### **Ввод-вывод 2-го типа**

Описанные выше буферы ввода и вывода отличаются тем, что запись, которая должна быть введена или выведена рабочей программой, физически перемещается относительно области памяти, являющейся стандартной для данной программы. Это перемещение выполняется либо до обработки (в случае ввода записи), либо после завершения обработки (в случае вывода записи). Именно поэтому операции по вводу начинаются сразу после извлечения последней записи из блока буфера или операции по выводу — сразу после помещения последней записи в блок буфера.

При буферизации 2-го типа операции по вводу не начинаются, пока программа ищет 1-ю запись следующего блока. Когда она находит эту запись, предполагается, что с предыдущей записью уже никакие операции производиться не будут, и ОС начинает операции по вводу. Это означает, что нет необходимости помещать запись в стандартное место в памяти, так как рабочая программа может извлечь ее прямо из блока буфера. Работа ОС по указанию очередной вводимой записи заключается просто в изменении базового регистра адреса или ячейки, являющейся началом следующей записи.

Подобным же образом для 2-го типа буфера вывода последний блок, который заполнен, не может быть переписан до тех пор, пока рабочая программа не укажет

местоположение записи, которая станет 1-й записью следующего блока. Когда это происходит, ОС считает, что с последней записью предыдущего блока больше никакие операции производиться не будут, и начинает операции по переписи заполненного блока в файл вывода. Как и в предыдущем случае, запись не перемещается физически, а выбирается прямо с того места, где она получена. Это означает, что следующая выводимая запись может быть сформирована в текущих ячейках блока вывода.

Операторы открытия и закрытия файлов, применяемые в буферах 1-го типа, используются для тех же целей и в буферах 2-го типа. Однако условия прерывания рабочей программы при буферизации 2-го типа отличаются от описанных условий для буферизации 1-го типа. После того как рабочая программа закончила обработку последней записи для блока буфера вывода 2-го типа, она должна указать ОС место, где будет получена 1-я выходная запись следующего блока буфера вывода. Если к этому моменту ОС определит, что для этого нового блока буфера операция вывода, начатая ранее, еще не закончилась, рабочая программа прерывается и выбирается новое СОТ. В случае буфера ввода 2-го типа рабочая программа обращается к ОС за указанием адреса следующей записи. Если при этом оказывается, что обращение должно произойти к первой записи следующего блока, а для него к этому времени еще не закончены операции по вводу, то рабочая программа прерывается, и в решение выбирается следующая программа.

Так же, как и при буферизации 1-го типа, рабочей программе не представляется возможности продолжать работу по другой ветви, если скорость обработки данных превышает скорость считывания блоков из файла ввода или скорость записи блоков в файл вывода.

### **Общие замечания**

Буферизация 2-го типа более эффективна в случаях, когда программа считывает входные записи, обрабатывает их и непосредственно выдает выходные записи. В этом случае система избегает промежуточной буферизации записей в стандартных ячейках, как это де-

ляется при буферизации 1-го типа. Но буферизация 2-го типа обладает и недостатками, снижающими ее эффективность. Это связано с тем, что система не может начать действия по вводу-выводу последнего блока буфера до тех пор, пока рабочая программа не предъявит следующей записи для файла вывода (или не запросит следующей записи для файла ввода). Иными словами, ОС не может начать действия по вводу-выводу текущего блока до тех пор, пока точно не будет известно, что все действия с последней записью этого блока уже закончены. Сигналом окончания работы с предыдущей записью является обращение к следующей записи файла, т. е. к первой записи следующего блока буфера. Это означает, что при буферизации 2-го типа действия по вводу-выводу не начинаются до тех пор, пока обработка предыдущей записи не закончится, в то время как при буферизации 1-го типа действия по вводу-выводу могут начаться во время обращения к последней записи и могут протекать параллельно с ее обработкой.

Итак, буферизация 1-го типа требует физического перемещения записей в памяти с соответствующими затратами времени ЦП, но зато допускает обработку последней записи блока с одновременным выполнением операций ввода-вывода для этого блока. При буферизации 2-го типа более эффективно используется ЦП за счет прямого обращения к записям в блоке без их физического перемещения в памяти. Но наряду с этим буферизация 2-го типа не допускает проведения операций ввода-вывода для блока до тех пор, пока полностью не закончится обработка последней его записи, о чем программа сообщает обращением к следующей записи.

Таким образом, при буферизации 1-го типа происходит снижение эффективности использования ЦП, а при буферизации 2-го типа — снижение эффективности использования периферийных устройств.

### **Запрос ввода-вывода**

Введем понятие *запроса ввода-вывода*. Оно будет в дальнейшем использоваться при описании ОС с буферами 1-го и 2-го типов. Назначение запроса ввода-вывода состоит в том, чтобы начать действия по вводу-выводу, необходимые при работе программы. Иными сло-

вами, когда рабочая программа выполняет оператор ввода-вывода, она создает запрос на ввод-вывод, который затем используется для реализации этих действий. Запрос ввода-вывода во многих отношениях сходен с запросом ЦП, введенным ранее, и состоит из *поля запроса* и *слова запроса*. Содержимое слова запроса является адресом поля запроса. Поле запроса ввода-вывода содержит всю необходимую информацию для выполнения операционной системой работы по вводу-выводу, запрошенной рабочей программой.

Обратим внимание на то, как рабочая программа обращается к файлу. Каждое выполнение этой программы (возможно, и повторное) связано с соответствующим набором буферов в памяти (по одному буферу на каждый файл). Тогда вся информация, касающаяся состояния буферов для рабочей программы, и другая информация, необходимая для получения программой записей из файла, должна содержаться в определенном месте памяти. Память, отведенную для этих целей, мы и называем *полем запроса ввода-вывода*. Слово запроса ввода-вывода, связанное с этим полем, делает работу операционной системы более удобной.

### **Возникновение запросов ввода-вывода**

Значение понятия запроса состоит в том, чтобы представить программу как элемент вычислительной системы, которой во время своего присутствия в системе требует лишь память для своего размещения. Тогда в этом смысле программа представляет собой «план» для использования ресурсов ВС во время выполнения этой программы. При нашем взгляде на ОС мы можем рассматривать запрос в качестве сущности, использующей ресурсы в соответствии с «планом» программы, которая выполняется или, скорее, через которую проходит запрос. С этой точки зрения мы можем сказать, что выполнение программы — это проход запроса через команды программы. Во время этого прохода различные команды могут либо производить вычисления, либо вызывать передачу запроса в ОС, либо создавать вторичные запросы, определенные выше как запросы ввода-вывода.

Таким образом, существует два типа запросов, с которыми имеет дело операционная система. Первым

является запрос ЦП, с которым читатель уже познакомился в предыдущих главах. Он является запросом на выполнение определенной рабочей программы. Для выполнения такой программы, которая обращается к буферируемым файлам, необходимо ввести второй тип запросов, называемых *запросами ввода-вывода*. Следует четко представлять себе, что они генерируются запросами ЦП в соответствии с «планом» программы, когда запрос ЦП проходит через такие команды, как: читать, писать, выдать на печать, и т. д.

Запрос ввода-вывода играет для ОС такую же важную роль, как и запрос ЦП. Запрос ЦП представлен полем запроса и связанным с ним словом запроса. Это допускает обработку операционной системой слова запроса, которое является представителем поля, связанного с ним, и, благодаря этому, слово является и представителем рабочей программы.

Аналогично и запрос ввода-вывода состоит из поля запроса и слова запроса. Слово запроса ввода-вывода доступно для обработки операционной системой и является представителем поля запроса, с которым оно связано. Далее, для различения этих двух типов запросов мы будем называть их соответственно запросом ЦП и запросом ввода-вывода. Кроме того, мы будем называть слова запросов соответственно «словом ЦП» и «словом ввода-вывода», если такое упрощение не будет вносить двусмысленности.

### Поле запроса ввода-вывода

На рис. 27 представлены запросы ЦП и ввода-вывода. На этом рисунке эти запросы собраны вместе. Соответствующие слова ЦП и ввода-вывода указывают на связанные с ними поля запросов. Так, слово ЦП указывает на начало поля запроса ЦП, состоящее в нашем случае из нескольких подполей.

Кроме того, на рисунке представлено два слова запроса ввода-вывода для двух файлов, связанных с этим запросом ЦП. Первое слово запроса ввода-вывода указывает на поле запроса 1-го файла, а второе — на поле запроса 2-го файла.

Понятно, что каждое слово ввода-вывода представляет текущее обращение к файлу, с которым оно свя-

зано через соответствующее поле ввода-вывода. Итак, в любое время может существовать несколько слов ввода-вывода, сгенерированных данным запросом ЦП. Каждое из этих слов представляет текущее обращение к записи из соответствующего файла. Более того, в любой момент может существовать несколько слов ввода-вывода, представляющих обращение к одному и тому

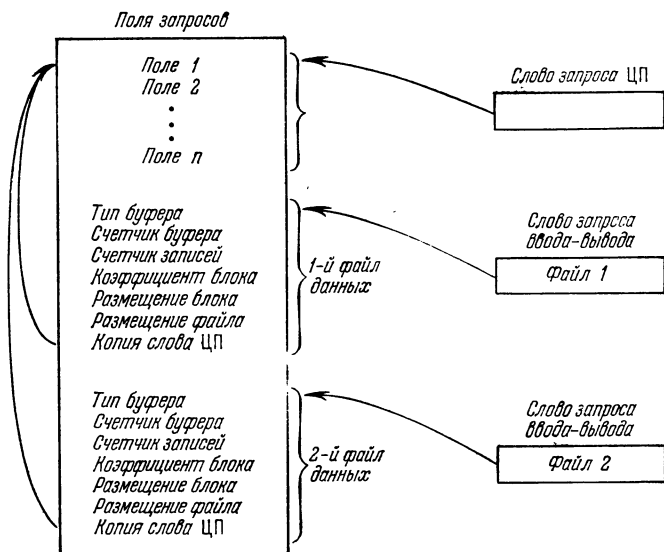


Рис. 27. Запросы ЦП и ввода-вывода.

же файлу. И, наконец, может существовать несколько слов ввода-вывода, одновременно обрабатываемых ОС и представляющих каждое из описанных выше обращений к различным файлам, связанным с различными запросами ЦП, одновременно обрабатываемыми системой.

Поле запроса ввода-вывода должно содержать информацию о каждом из файлов, которые могут понадобиться запросу ЦП при его проходе через программу. Минимум информации, необходимой ОС для управления буферизацией, приведен на рис. 27. *Тип буфера* — это код, который указывает, является ли данный буфер буфером ввода или вывода, а также буфером 1-го или 2-го типа.



*Счетчик буфера* первоначально указывает полное число блоков, связанных с данным файлом во время работы определенной программы. Если блок буфера ввода опустошается программой, счетчик буфера уменьшается на единицу. При пополнении этого блока счетчик увеличивается на единицу. Аналогично, если блок буфера ввода заполняется программой, счетчик уменьшается на единицу и увеличивается на единицу при передаче информации из этого блока на периферийное устройство, на котором находится данный файл. Таким образом, счетчик буфера указывает число блоков, которые рабочая программа может одновременно использовать, включая и те блоки, записи которых находятся в обработке. Из-за того, что управление буфером, описываемое далее, зависит от запросов, число блоков буферов, связанных с каждым независимым выполнением данной программы, может быть различным.

*Счетчик записей* применяется для текущего блока буфера ввода и представляет собой число еще необработанных записей. Для буфера вывода этот счетчик представляет число незанятых позиций под записи.

*Коэффициент блока* уже определялся ранее и означает число записей в блоке. Он используется операционной системой для восстановления счетчика записей.

*Размещение блока* — это адрес первой ячейки первой записи блока. Предполагается, что остальные блоки буфера либо образуют в памяти непрерывную последовательность, либо порядок их следования устанавливается при помощи адресных связей внутри самого буфера, либо порядок следования задается путем перечисления адресов блоков в поле запроса для данного файла.

*Размещение файла* — это информация о местоположении файла на периферийном устройстве.

Назначение запроса ввода-вывода служит удобным средством для представления работы по вводу-выводу, происходящей параллельно с обработкой вызвавшего его запроса ЦП. Это означает, что запросы ввода-вывода должны согласовываться по времени с их запросами ЦП. *Копия слова ЦП* представляет собой удобное средство для этого согласования.

Поле запроса ввода-вывода заполняется одновременно с заполнением поля запроса ЦП. Это происхо-

дит тогда, когда введен заказ на выполнение программы, и из него извлечена (директором) вся необходимая информация для заполнения поля запроса ввода-вывода. Следовательно, поле запроса ввода-вывода связано с каждым выполнением данной программы и, конечно, с числом блоков буфера для данного выполнения программы. Слова ввода-вывода генерируются динамически, когда запрос ЦП встречает оператор ввода-вывода в программе, через которую он проходит. Итак, при каждом выполнении оператора ввода-вывода запросом ЦП создается слово запроса. Содержимое этого слова является адресом информации в поле запроса ввода-вывода, относящейся к файлу, к которому произошло обращение из программы.

### Расширение поля запроса ЦП

Поле ЦП определялось ранее как поле, состоящее из 2-х слов, помещенных в фиксированных ячейках памяти, доступных для ОС. Первым словом был текущий обрабатываемый запрос (или ячейка COT). Это слово содержало слово запроса ЦП (т. е. адрес поля запроса, которому предоставлялся в данное время процессор). Вторым словом поля ЦП было слово готового к обработке запроса (AT). Это слово использовалось ОС для удобства управления мультипрограммным режимом. Для того чтобы дать возможность обработки запросов ввода-вывода, поле ЦП расширяется, чтобы включить третье слово, которое будет содержать слово запроса ввода-вывода, обрабатываемого в настоящий момент операционной системой. Это слово будем обозначать IOT в поле ЦП.

В качестве примера использования ячейки IOT поля ЦП рассмотрим случай, когда запрос рабочей программы *W*, работающий в качестве COT, выполняет оператор READ (читать) в рабочей программе. При этом создается, во-первых, соответствующее слово запроса ввода-вывода, содержимое которого — это адрес информации о файле в поле запроса ввода-вывода. Это слово мы будем обозначать *w*. Вторым действием, совершаемым запросом *W*, является передача управления операционной системе. При этом запрос операционной системы *Z*

становится текущим обрабатываемым запросом, а запрос работы  $W$  становится готовым к обработке запросом. Следовательно, при переходе в ОС поле ЦП имеет в ячейке COT слово запроса  $Z$ , в ячейке AT — слово

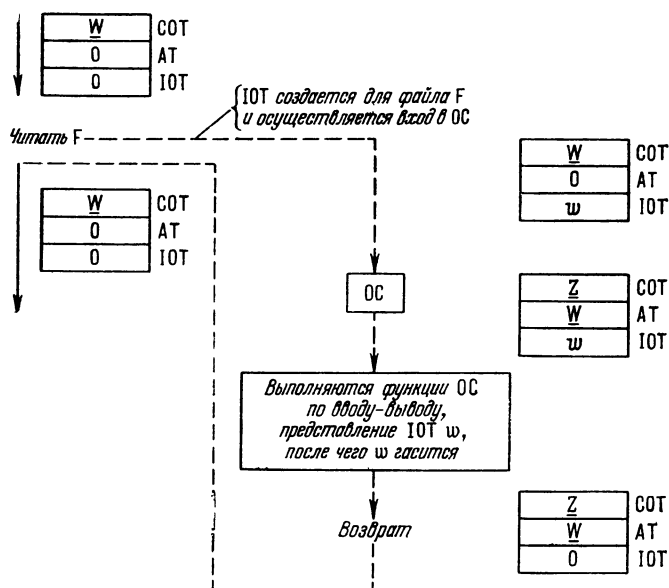


Рис. 28. Использование ячейки IOT.

запроса рабочей программы  $W$ , а в ячейке IOT — сгенерированное слово запроса ввода-вывода  $w$ . Эти два последних слова являются готовыми к обработке для текущего запроса операционной системы  $Z$ . Рис. 28 подводит итог этим действиям, которые генерируют слово запроса ввода-вывода в ячейке IOT поля ЦП.

В следующих разделах мы опишем функцию буфера подробнее. Она заключается в том, чтобы распознать состояние соответствующего буфера ввода-вывода и произвести необходимые операции с файлом. Информация о файле во время работы буфера представлена словом запроса ввода-вывода, помещенным в ячейку IOT поля ЦП (в главе 9 мы опишем модификацию состояний ЦП для диаграмм перехода поля ЦП).

## Состояния буфера ввода первого типа

В нашем дальнейшем описании мы часто будем ссылаться на *модуль состояния буфера*, назначение которого определять состояние буфера при обращении по запросу ввода-вывода.

Для каждого из рассмотренных до сих пор буферов, для которых мы ищем методы управления, существует 3 различных состояния. Рассмотрим сначала буфер ввода 1-го типа. Если программа хочет получить входную запись, которая не является последней записью текущего блока буфера, мы будем обозначать такое состояние буфера А. Если программа пытается получить последнюю запись из текущего блока буфера, мы будем говорить, что буфер находится в состоянии В. Если программа ищет первую запись блока буфера, а пополнение этого блока еще не закончено, мы будем говорить, что буфер находится в состоянии С. Такие же состояния существуют и для буфера вывода 1-го типа и для буфера ввода-вывода 2-го типа.

Так как деятельность по вводу-выводу возложена на операционную систему, то она должна обладать способностью определять состояние буфера для каждого запроса ввода-вывода.

### Буфер ввода первого типа

Рассмотрим логические действия, которые выполняются модулем состояния буфера ввода 1-го типа для того, чтобы определить текущее состояние буфера ввода, связанного с данным запросом ввода-вывода. Пусть имеется запрос, слово которого находится в ячейке IOT поля ЦП. Мы будем называть его запросом ввода-вывода *w*. На рис. 29 приведена блок-схема логики выполнения модуля состояния буфера.

Текущий запрос попадает в блок 1, где проверяется счетчик буфера, находящийся в поле запроса ввода-вывода. Если он отличен от нуля, управление передается в блок 2, где счетчик записей уменьшается на единицу. В блоке 3 извлекается очередная запись из блока буфера и передается программе, связанной с *w*. Таким образом, число, содержащееся в счетчике, представляет собой число записей, остающихся в те-

кущем блоке буфера. В блоке 4 проверяется, была ли извлеченная запись последней в блоке. Если нет, то в блоке осталась по крайней мере одна запись. Буфер находится в состоянии А. При этом осуществляется соответствующий этому состоянию выход.

Предположим теперь, что после извлечения очередной записи блок буфера стал пустым. В этом случае буфер находится в состоянии В. Происходит указанный

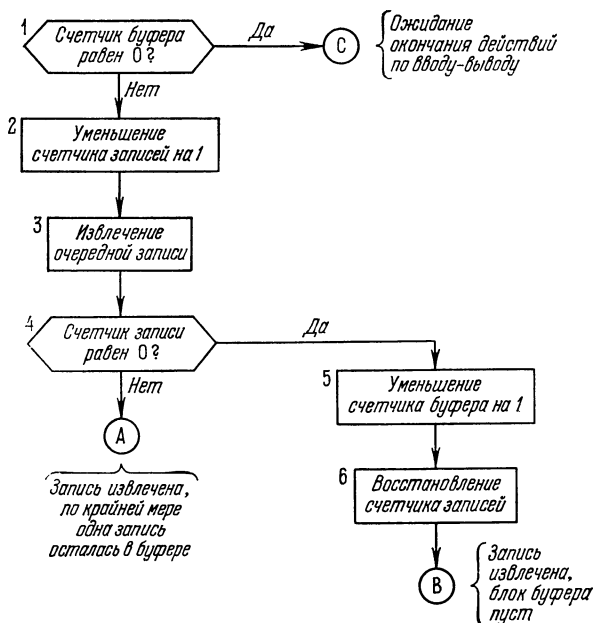


Рис. 29. Функция буфера ввода 1-го типа.

на блок-схеме выход. Однако перед этим необходимо уменьшить на единицу счетчик буфера (он равен числу блоков буфера, доступных для обработки), а также восстановить счетчик записей по значению коэффициента блока. Это означает, что при следующем обращении к буферу будет извлекаться первая запись из следующего блока.

Еще один выход соответствует такому состоянию буфера, при котором по данному  $w$  невозможно извлечь информацию из буфера. Вернемся к блоку 1. Такая ситуа-

ция отражается на значении счетчика буфера для данного запроса  $w$ . Значение этого счетчика равно нулю. Это означает, что буфер пуст. При этом осуществляется указанный на рис. 29 выход.

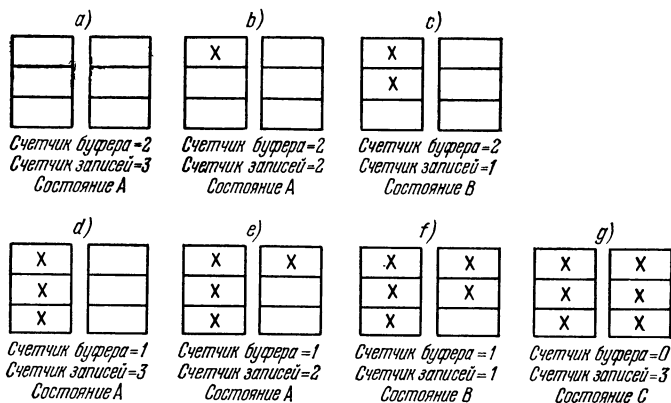


Рис. 30. Иллюстрация работы буфера.

Для пояснения работы модуля состояния буфера на рис. 30 приведен пример буфера, имеющего 2 блока с коэффициентом блока, равным 3. Рассматривается буфер ввода 1-го типа. Сначала (рис. 30, а) в каждом блоке содержится по 3 записи. Счетчик буфера равен 2, счетчик записей равен 3. Каждый из них используется тогда, когда рабочий запрос обращается к буферу. Указанное на рис. 30 состояние буфера определяется при выходе из модуля состояния буфера.

Рис. 30, а представляет ситуацию, когда к моменту запроса из буфера еще не выбрана ни одна запись. Следовательно, буфер находится в состоянии А. Затем, при повторном обращении к модулю состояния буфера данного запроса ввода-вывода, счетчик записей уменьшился на единицу, а счетчик буфера не изменился (рис. 30, б). Следовательно, буфер все еще остается в состоянии А. Однако на рис. 30, с ситуация изменяется, так как счетчик записей уменьшился на единицу, указывая тем самым, что при новом обращении из блока будет выбираться последняя запись. Следовательно, буфер переходит в состояние В. При следующем обращении к модулю состояния буфера со стороны запро-

са ввода-вывода (рис. 30, *d*) счетчик записей восстанавливается по значению коэффициента блока и становится равным 3. Выборка записей будет теперь производиться из второго блока буфера, и при этом может производиться (операционной системой) пополнение 1-го блока буфера. Следующая запись выбирается из 2-го блока и не опустошает его. Следовательно, состояние буфера А (рис. 30, *e*) не меняется. Однако при выборке следующей записи значение счетчика записей при входе в модуль состояния буфера уменьшилось до единицы, т. е. далее будет выбираться последняя запись из блока. Буфер перейдет в состояние В (рис. 30, *f*).

На рис. 30, *d* изображена ситуация после выхода из состояния В. Значение счетчика буфера равно нулю, а счетчик записей, восстановленный по значению коэффициента блока, равен 3. Если к моменту обращения программы за следующей записью операции по пополнению 1-го блока буфера не окончатся, буфер окажется в состоянии С. Однако, если эти операции закончатся к этому моменту, то счетчик буфера увеличится на единицу и буфер перейдет в состояние А.

### **Буфер вывода первого типа**

На рис. 31 приведена блок-схема работы модуля состояния буфера вывода 1-го типа. Сравнивая его с рис. 29, можно видеть, что они весьма сходны. Более того, они отличаются всего одной операцией, а именно, работой блока 3, в котором запись, полученная из программы, помещается в буфер для вывода. Так как отличие между вводом и выводом зависит от определения буфера, это отличие отражается в поле запроса ввода-вывода. Таким образом, модуль состояния буфера ввода и буфера вывода 1-го типа может быть оформлен в виде одной программы, которая производит операции по вводу или выводу согласно информации, содержащейся в поле запроса ввода-вывода.

### **Буфер второго типа**

Напомним, что буфер 2-го типа дает возможность обрабатывать запись на месте без перемещения ее из буфера в специально отведенное место памяти. Различие между двумя типами буферизации сказывается во время

определения состояния буфера. Говорят, что буфер находится в состоянии А, если в начале работы модуля состояния буфера ввода 2-го типа в блоке буфера есть по крайней мере одна запись. Буфер находится в состоянии В, если в начале работы модуля состояния буфера

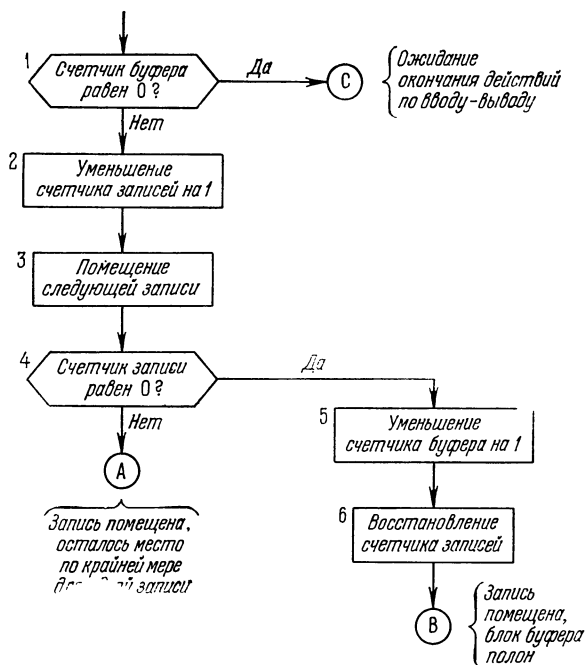


Рис. 31. Функция буфера вывода 1-го типа.

в блоке буфера нет записей. И, наконец, буфер находится в состоянии С, если в начале работы модуля состояния буфера следующая запись не может быть получена, так как она является первой записью следующего блока, а он еще не заполнен.

Эти три состояния буфера одинаково справедливы и в случае буфера вывода 2-го типа и, как мы увидим несколько позже, для этих 2-х типов буферов можно использовать одну и ту же программу. Сейчас мы приходим к описанию модуля состояния буфера ввода 2-го типа. Блок-схема работы модуля состояния буфера ввода 2-го типа приведена на рис. 32.



Счетчик буфера, счетчик записей и коэффициент блока используются для определения текущего состояния буфера и для его изменения. Работа модуля состояния буфера начинается в блоке 1. Как и в предыдущем описании буферизации 1-го типа, модуль состояния буфера начинает свою работу по запросу ввода-вывода  $w$ . Итак, первым действием, предпринимаемым модулем состояния буфера, является проверка значения счетчика записей.

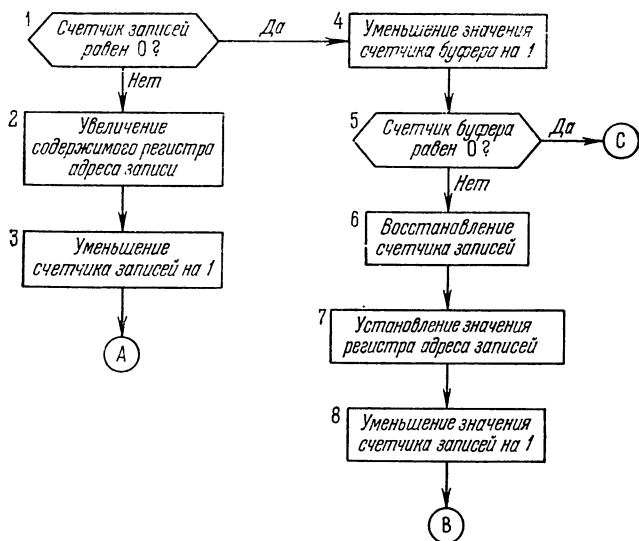


Рис. 32. Функция буфера ввода 2-го типа.

Предположим, что значение его отлично от нуля. Тогда в блоке 2 значение регистра адреса записи увеличивается так, что теперь, оно становится равным адресу следующей записи. В блоке 3 значение счетчика записей для  $w$  уменьшается на единицу. Для данного запроса ввода-вывода буфер находится в состоянии А, и осуществляется соответствующий выход. Должно быть понятно, что текущее значение счетчика записей для данного запроса ввода-вывода указывает число записей, еще не обработанных программой.

Если значение регистра адреса записей таково, что указывает на последнюю запись в блоке буфера, то в блоке 3 значение счетчика записей становится равным

нулю. Это означает, что при следующем обращении к модулю состояния буфера для данного  $w$  значение счетчика записей равно нулю и из блока 1 мы попадаем в блок 4. Здесь значение счетчика буфера уменьшается на единицу. Предполагается, что первоначально в этом счетчике содержится полное число блоков, выделенных под буфер для данного  $w$ . В любой момент времени значение этого счетчика указывает число блоков буфера, еще не обработанных программой, включая и блок, находящийся в данный момент в обработке. В блоке 5 производится проверка значения счетчика буфера. Предположим, что оно отлично от нуля. В этом случае в блоке 6 происходит восстановление счетчика записей.

Восстановление счетчика записей производится по значению коэффициента блока. Значение регистра адреса записей для данного запроса становится таким, чтобы давать адрес первой записи следующего блока буфера (блок 7). В блоке 8 значение счетчика записей уменьшается на единицу, указывая тем самым, что в блоке стало на одну запись меньше. Буфер находится в состоянии В, что и указано на рис. 32.

Последним условием, при котором работа модуля определения буфера может быть продолжена, — это состояние С для буфера. Последняя запись последнего блока буфера должна быть обработана программой, и значение счетчика записей станет равным нулю. При последнем обращении к модулю определения буфера управление из блока 1 перейдет в блок 4, а затем в блок 5, где и будет установлено, что счетчик записей равен нулю. Будет сделан соответствующий состоянию С выход из модуля состояния буфера.

Основное отличие между буфером ввода 2-го типа и буфером вывода 2-го типа заключено в выполнении работы по вводу-выводу в связи с состоянием В буферов. В первом случае требуются операции ввода, в то время как во втором случае — операции вывода. Остальные же операции совпадают. Следовательно, логика работы этих буферов не зависит от направления перемещения данных между буфером и программой.

Проблема отличия буфера ввода от буфера вывода 2-го типа решается путем обращения к полю запроса ввода-вывода, связанного с данным словом ввода-вывода.

## Заключение

Эта глава была предназначена для тех, кому необходимо расширить возможности ОС, с целью получить возможность буферизации ввода-вывода при работе программ. Основным критерием было закрепление за ОС управления буферизацией. Этот критерий появился в результате разработки основной идеи, выраженной в главе 3, а именно: заставить рабочую программу приостановиться, если скорость ее работы превышает возможности буфера для одного из ее файлов.

Имея это в виду, мы пришли к расширению понятия поля ЦП так, чтобы охватить запросы ввода-вывода и дать ОС возможность обработки слов запроса ввода-вывода. Кроме этого, было разработано понятие модуля состояния буфера, который стал основным орудием при обеспечении операционной системы возможностью буферизации. Следует отметить, что модуль состояния буфера является просто средством для достижения этой цели, так же как и машинные команды являются средством для написания программ. Тогда в этом смысле модуль состояния буфера выполняет специфические, точно определенные действия и может быть использован в любом контексте так, чтобы не нарушать логики данных операций. Таким образом, модуль состояния буфера может рассматриваться в качестве оператора, которым может пользоваться операционная система в необходимых случаях. Однако условия для применения этого модуля состояния зависят от конкретной реализации ОС.

## ГЛАВА 9

### МОДЕЛЬ МОНИТОРА (МРХ)

#### Введение

В этой главе мы рассмотрим монитор для обработки с буферизацией рабочих программ, работающих в мультипрограммном режиме. Этот МРХ является моделью в том смысле, что он выполняет требование на ввод-вывод для буферизируемого ввода-вывода так, как это описано в главе 3. Основным средством для этого служит описанный в главе 8 модуль состояния буфера, работа которого характеризует состояние буфера ввода-вывода для каждого данного запроса в терминах выбора одного из трех возможных выходов из этого модуля. Эти выходы описывают условие буфера, но только предполагают действия, которые могут быть предприняты монитором, а не выполняют их. Таким образом, модель, описываемая в настоящей главе, разрабатывается, чтобы реализовать эти действия. В широком смысле мониторы, присутствующие в проблемно-ориентированных ОС, содержащих блоки приема и вывода информации, могут существенно отличаться друг от друга.

#### Описание состояния ЦП

Ранее мы описали выражение, определяющее состояние ЦП как такую пометку на блок-схеме, которая указывает, какой в данный момент запрос является запросом ЦП, а какой является готовым к обработке запросом. Это выражение имело вид  $A(B)$ , где  $A$  — имя текущего обрабатываемого запроса (или COT), а

В — имя готового к обработке запроса (или АТ). Далее говорилось, что имеющийся запрос обрабатывается в СОТ тогда, когда запрос появляется в поле АТ. Оба запроса СОТ и АТ являются запросами центрального процессора. В поле центрального процессора эти 2 запроса представлены в виде 2-х ячеек, обозначенных СОТ и АТ.

В предыдущей главе поле ЦП было расширено для включения в него еще одной ячейки — слова запроса ввода-вывода (ячейка ИОТ). Как указывалось в этой же главе, ИОТ служит для хранения слова запроса ввода-вывода, и, когда в СОТ находится Z, выполняется модуль состояния буфера. Поэтому мы переопределим выражение для состояния ЦП так, чтобы включить в него ИОТ. Теперь форма, которую будет иметь выражение для состояния ЦП, станет такой:  $A (B, c)$ . Здесь:  $A$  и  $B$  (как и ранее) — имена запросов ЦП, т. е. слова запросов ЦП, а  $c$  — имя запроса ввода-вывода, обрабатываемого СОТ.

### Операторы перехода

Перейдем теперь к описанию шестнадцати операторов перехода, используемых при описании работы МРХ с буферизацией. Как будет видно позднее, эти операторы обладают характеристиками операторов языка программирования.

Их синтаксическая структура в основном такова: строка алфавитных символов, за которой следует параметр, заключенный в скобки. Вслед за ним следует строка символов, за которой следует параметр, заключенный в скобки.

Параметры, встречающиеся в этом определении, состоят в основном из одного или двух названий, разделенных наклонной чертой, представляющей альтернативный выбор. Кроме того, каждое из названий будет представлять данные, находящиеся либо в поле ЦП, либо в поле запроса ЦП, либо в поле запроса ввода-вывода. Разделение этих названий по трем категориям изображено на рис. 33.

Например, если копия слова ЦП используется в качестве описания параметра в одном из этих операторов, значит, мы имеем дело с полем запроса ввода-вывода и,

следовательно, этот оператор должен использоваться с соответствующим словом ввода-вывода в ИОТ поля ЦП.

Эта формальность вводится для того, чтобы избежать двусмысленности при описании блок-схем. Так

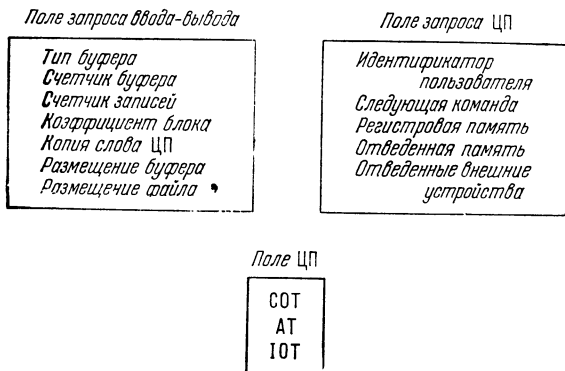


Рис. 33. Запросы ЦП и ввода-вывода.

как работа монитора заключается, в основном, в обработке слов ЦП и ввода-вывода, мы увидим, что этих операторов достаточно для того, чтобы дать точное представление о логике МРХ.

### Буфер (один/два)

Этот оператор вводится вместе со словом ввода-вывода в ячейку ИОТ поля ЦП. При обращении к полю запроса ввода-вывода, которое производится с помощью слова запроса ввода-вывода, выполняется модуль состояния буфера, описанный в главе 8. Выход А из этого модуля говорит о том, что запись, запрашиваемая запросом ввода-вывода, доступна и что есть еще по крайней мере одна запись в блоке буфера. Выход В говорит о том, что очередная запись доступна, но она является последней в этом блоке буфера. Выход С говорит о том, что запись недоступна по данному запросу ввода-вывода. Параметр *один* выбирается тогда, когда работает модуль состояния буфера 1-го типа, а параметр *два*, когда — буфер 2-го типа.

### **Копировать (очередь/имя) в (очередь/имя)**

Содержимое указанной ячейки очереди или ячейки памяти копируется в соответствующую ячейку очереди или памяти. Ячейками очередей являются те ячейки, на которые указывают счетчики очередей. Если используется параметр *имя*, то с помощью него можно обратиться к АТ, ИОТ или к любой другой символической ячейке. Если в качестве параметра этой операции не используются АТ или ИОТ, то его работа не влияет на состояние ЦП. Если же в качестве параметра используются атрибуты полей запросов ЦП или ввода-вывода, то с помощью него можно влиять на содержимое ячеек АТ или ИОТ в полях запросов.

### **Цикл**

В реальной машине этот оператор является оператором перехода и представляет собой точку ожидания. Оператор дает также возможность прерывания по окончании операций ввода-вывода. Таким образом, если СОТ вошел в этот оператор, то выйти из него можно только по прерыванию. Вообще, содержимое поля ЦП может быть любым при входе в этот оператор. Однако, как будет показано на примере, выход из оператора по прерыванию окончания ввода-вывода может испортить содержимое ячейки АТ.

### **Уничтожить (очередь/имя)**

При выполнении данного оператора содержимое указанной ячейки обнуляется. Если параметром оператора является поле запроса ЦП или ввода-вывода, выражение состояния ЦП при входе в этот оператор должно содержать ячейки АТ или ИОТ.

### **Увеличить счетчик буфера**

В этот оператор можно войти со словом запроса ввода-вывода в ячейке ИОТ поля ЦП. При выполнении этого оператора счетчик буфера, которому соответствует слово ввода-вывода в ячейке ИОТ, увеличивается на единицу.

### **Сравнить (АТ/ІОТ) с (очередь/имя)**

Если в качестве параметра указано название очереди, тогда каждый элемент этой очереди, начиная с первого, сравнивается с содержимым ячейки АТ или ІОТ поля ЦП. Из этого оператора существует два выхода. Если пара одинаковых названий не подобрана, тогда происходит первый. Если же пара подобрана, тогда происходит второй выход, и в этом случае указатель очереди устанавливается напротив элемента, которому подобрана пара (АТ или ІОТ). Если в качестве параметра используется имя, то происходит сравнение с содержимым указанной ячейки. Если параметр указывает на ячейку из поля запроса ЦП или ввода-вывода, то оператор обратится к содержимому ячейки АТ или ІОТ в выражении состояния ЦП.

### **Режим ОС**

Этот оператор уже был введен ранее. Его функция состоит в том, чтобы поместить СОТ в ячейку АТ и восстановить запрос операционной системы Z, запоминая его в ячейке СОТ. Содержимое ячейки ІОТ остается нетронутым. Кроме того, дальнейшие прерывания ввода-вывода запрещаются. Предполагается, что аппаратура запоминает прерывания, возникающие во время их запрета. При первой возможности система обрабатывает эти прерывания.

### **Поместить (АТ/ІОТ) в (очередь/имя)**

В этот оператор входят либо со словом ЦП, либо со словом ввода-вывода, помещенными в ячейках АТ или ІОТ, как описано первым параметром. Функция этого оператора состоит в том, чтобы поместить соответствующее слово из поля ЦП в очередь или ячейку памяти. Ячейка, из которой выбирается слово, обнуляется. Если параметр является именем поля запроса ЦП или поля запроса ввода-вывода, тогда выражение состояния ЦП при входе в этот оператор должно содержать соответственно АТ или ІОТ в качестве имеющегося запроса. В этом случае ячейки АТ или ІОТ обнуляются.



### **Поместить IOT в (очередь) к каналу**

В этот оператор входят со словом ввода-вывода, находящимся в ячейке IOT поля ЦП. Это слово перемещается в соответствующую очередь, являющуюся общей для всех очередей, связанных с каналами ввода-вывода. Так, канал 1 может быть связан с IO1 (очередь 1), канал 2 — с очередью IO2 и т. д. Ячейка IOT оператором обнуляется.

### **Возврат (COT/AT)**

Оператор возврата уже описывался ранее. Если используется параметр COT, то управление передается программе пользователя, которая была последней в COT. Если используется параметр AT, то управление передается программе пользователя, готовой к обработке. В первом случае ячейки AT и IOT поля ЦП не изменяются. Однако во втором случае содержимое ячейки COT заменяется содержимым ячейки AT, а сама ячейка AT обнуляется. Содержимое ячейки IOT не изменяется. Кроме того, разрешаются прерывания по окончанию ввода-вывода.

### **Выбрать (AT/IOT) из (очередь/имя)**

Если в качестве параметра используется название очереди, то оператор выбирает первый элемент из этой очереди и помещает его либо в AT, либо в IOT. Если в качестве параметра используется имя, то осуществляется непосредственная пересылка. В любом случае содержимое ячеек AT или IOT заменяется выбранным словом запроса. Вместо выбранного слова в очередь или в ячейку памяти засылается ноль.

Если очередь пуста или содержимое указанной ячейки равно нулю, осуществляется специальный выход из этого оператора.

### **Выбрать IOT из (очередь) к каналу**

Предполагается, что номер канала определяется по прерыванию и используется оборудованием при выполнении этого оператора. Одна из нескольких очередей,

связанных с набором каналов, является источником слов запроса ввода-вывода. Выбор слова делается в соответствии с критерием, установленным для очереди. Содержимое ячейки IOT поля ЦП заменяется выбранным словом ввода-вывода, а его место в очереди заменяется нулем.

### **Начать ввод-вывод**

В этот оператор входят со словом запроса ввода-вывода в ячейке IOT поля ЦП. По этому оператору начинается передача данных между буфером и периферийным устройством.

### **Перевести указатель (очередь)**

По этому оператору указатель соответствующей очереди перемещается к следующему элементу.

### **Проверить тип буфера**

В этот оператор входят со словом ввода-вывода в ячейке IOT поля ЦП. При обращении к указанному полю запроса ввода-вывода определяется тип буфера для файла, связанного с этим запросом ввода-вывода. Из этого оператора существует 2 выхода. Первый выход соответствует буферу 1-го типа, второй — буферу 2-го типа.

### **Проверить IOT (канал/устройство)**

В этот оператор входят со словом ввода-вывода в ячейке IOT поля ЦП. Проверяется возможность использования канала или устройства, выделенного для данного слова ввода-вывода. Из этого оператора существует два выхода: первый соответствует случаю, когда канал или устройство свободны, второй — когда заняты.

### **Модель монитора**

Логика работы монитора, управляющего выполнением буферизуемых программ, приведена на блок-схемах рис. 34 и 35. В MPX имеется два входа. Первый

является входом по запросу на ввод-вывод, второй — по прерыванию, возникшему при окончании ввода-вывода.

### **Запрос ввода-вывода**

Когда рабочий запрос обрабатывает запрос на ввод-вывод, происходит обращение к ОС, которая определяет тип буфера, а затем состояние буфера, выделенного для операций ввода-вывода по данному запросу. Если следующая запись доступна для рабочей программы, то происходит выход из ОС.

Если следующая запись доступна для рабочей программы и запись эта является последней в данном блоке, тогда делается попытка инициирования запроса ввода или вывода этого буфера. Если попытка оказывается успешной, тогда происходит возврат к запросу рабочей программы. Если операции ввода-вывода не могут быть выполнены в этот момент, тогда слово запроса ввода-вывода помещается в соответствующую очередь и происходит возврат к запросу рабочей программы. В случае отсутствия доступных записей в блоке буфера, к которому происходит обращение, предполагается, что до этого была начата работа по вводу-выводу этого буфера. Дальнейшая обработка запроса рабочей программы прерывается. Отложенный запрос остается в этом состоянии до тех пор, пока не закончится работа по вводу-выводу буфера, связанного с этим запросом. По завершении операций по вводу-выводу этот запрос снова может поступить в обработку. Если запрос рабочей программы отложен по этой причине, необходимо, чтобы MPX выбрал какой-либо другой запрос из очереди AQ в качестве текущего обрабатываемого запроса.

Логика обработки монитором запросов ввода-вывода показана на блок-схеме рис. 34. В качестве СОТ выступает W. Рабочий запрос делает запрос на ввод-вывод. Его обработка оператором ввода-вывода дает в результате слово ввода-вывода, как указано на входе в блок 1.

В блоке 1 происходит определение типа буфера. В зависимости от результата происходит переход либо в блок 2, либо в блок 13. Никаких изменений в выражении ИП при этом не происходит.



Блоки 2 и 13 производят определение состояния буфера, к которому происходит обращение при помощи ИОТ. Если буфер находится в состоянии А, происходит переход в блок 3.

При входе в блок 3 уже известно, что следующая запись текущего блока буфера доступна (в случае буфера 1-го типа запись должна быть извлечена или помещена). При этом существует по крайней мере одна или более записей в этом блоке, доступных для обработки. Следовательно, запрос ЦП *W* может продолжать работу. Однако в это время ИОТ, выполнивший свою функцию, уже не нужен. Поэтому блок 3 должен уничтожить *w* путем обнуления ячейки ИОТ в поле ЦП. Это указано на выходе из блока 3.

Так как *W* является текущим обрабатываемым запросом (входа в ОС не происходило), оператор в блоке 4 переходит в СОТ, предполагая тем самым обработку ЦП запроса *W*.

В случае, если в блоке 2 или 13 определено, что буфер, к которому происходит обращение, находится в состоянии В, осуществляется переход в блок 5. Тем самым осуществляется вход в ОС, что изменяет поле ЦП. Это указано на выходе из блока 5. *Z* заменяет *W* в качестве СОТ, *W* становится готовым для обработки запросом, а *w* остается в качестве ИОТ. Прерывания по окончанию запрещаются.

Оператор блока 6 проверяет канал, связанный с буфером, к которому происходит обращение. Если канал не занят, то происходит переход в блок 7, где проверяется возможность доступа к устройству, на котором находится файл, необходимый для запроса. Если устройство также свободно, то происходит переход в блок 8, где начинается работа по вводу-выводу.

Заметим, что все эти действия не изменяют выражения ЦП. Следовательно, при входе в блок 9 при начале операций ввода-вывода слово запроса ввода-вывода помещается в список обращений в соответствии с выделенным каналом. Именно из этого списка и будет выбрано слово ввода-вывода, когда придет сигнал об окончании обмена в этом канале.

Часть функций блока 9 состоит в том, чтобы обнулить ячейку ИОТ, раз ИОТ удален из поля ЦП. Это указано на выходе из блока 9.

К этому моменту очередная запись стала доступной для запроса  $W$ . Эта запись извлекается (или помещается) в блок буфера, выделенного для  $W$ , и, как описывалось выше, начинается передача данных между буфером и периферийным устройством. Следовательно, здесь происходит возврат к запросу  $W$  для продолжения его работы. При этом разрешаются прерывания по окончании.

Вернемся теперь к блокам 6 и 7. Предположим, что либо канал, либо устройство для ИОТ ( $w$ ) заняты. Тогда происходит переход в блок 11. В этом случае невозможно начать передачу данных, и поэтому в блоке 11 ИОТ помещается в очередь, соответствующую данному каналу. Из этой очереди запрос может быть выбран после освобождения данного канала. При этом происходит обнуление ячейки ИОТ поля ЦП, как указано на выходе из блока 11.

К этому моменту запрос ввода-вывода от рабочего запроса  $W$  поставлен в известность о том, что запись получена (или помещена) и что запрос ввода-вывода  $w$  поставлен в соответствующую очередь. Он остается в этой очереди до тех пор, пока не поступит сигнал об окончании обмена в канале. После этого делается попытка начать следующий обмен. После выполнения этих действий может продолжаться работа запроса  $W$ . Это выполняется в блоке 12 путем передачи управления имеющемуся запросу.

Вернемся к блоку 2 (или 13) и предположим, что очередная запись в блоке буфера, выделенного ИОТ, недоступна. В этом случае буфер находится в состоянии С, и происходит передача управления в блок 14.

В блоке 14 происходит вход в ОС, и поэтому рабочий запрос  $W$  заменяется запросом операционной системы  $Z$  в качестве СОТ. Запрещаются прерывания по окончании ввода-вывода,  $W$  становится готовым к обработке запросом, и ИОТ не изменяется.

Теперь МРХ должен отложить дальнейшее использование запроса ввода-вывода  $w$  и дальнейшую обработку рабочего запроса  $W$ . Так как в настоящий момент нет записей, доступных для обработки в буфере, связанном с ИОТ, можно предположить, что предыдущий запрос ввода-вывода нашел этот буфер в состоянии В и согласно описанным выше действиям начал деятельность по

обмену, которая к настоящему моменту еще не закончилась.

Передача данных либо происходит в данный момент, либо запрос ввода-вывода стоит в соответствующей очереди. Поэтому в блоке 15 IOT, представляющем текущее требование буфера, запрос ввода-вывода помещается в очередь отложенных запросов DIOTQ, где он будет ожидать сигнала об окончании обмена в данном канале. По получении такого сигнала слово запроса ввода-вывода будет выбрано из DIOTQ и использовано для завершения текущего запроса ввода-вывода для запроса W.

Текущий запрос ввода-вывода, представляющий передачу данных между устройством и буфером, будет использоваться в дальнейшем, когда эта передача будет завершаться. Этот запрос будет использоваться для восстановления копии слова ЦП по содержимому поля запроса ввода-вывода, чтобы с ним можно было продолжить работу. Это и является причиной для сохранения копии исходного слова запроса ЦП в поле запроса ввода-вывода (см. рис. 33).

Таким образом, можно не сохранять обрабатываемый запрос W, так как его копия имеется в поле запроса ввода-вывода, где она обозначена как *копия слова ЦП*. В блоке 16 уничтожается АТ, и выражение ЦП становится таким, как указано на выходе из этого блока.

К этому моменту IOT и исходный запрос, по которому он был создан, откладываются. Задачей МРХ теперь является выбор следующего СОТ из числа ищущих доступ к ЦП. Эти запросы находятся в очереди АQ. Выбор очередного запроса является функцией блока 17. Если очередь АQ не пуста, то на выходе из блока 17 выбранный запрос появляется в качестве АТ в поле ЦП, что и указано на блок-схеме. Обработка выбранного запроса продолжается в блоке 18 при помощи оператора возврата.

В случае, если очередь АQ пуста, управление из блока 17 передается в блок 19. Отметим, что в это время не происходит никаких изменений в выражении ЦП, благодаря тому, что в очереди АQ отсутствуют запросы. В блоке 19 осуществляется вход в оператор CYCLE (цикл). Он разрешает прерывания по оконча-

нии ввода-вывода. Центральный процессор после этого остается в состоянии ожидания до тех пор, пока не придет прерывание по окончании ввода-вывода и не освободит отложенный запрос рабочей программы, поместив его в очередь AQ. Когда это произойдет, запрос операционной системы Z может продолжить работу MPX, дойдя до выбора запроса из очереди AQ.

### **Прерывание по окончании ввода-вывода**

Прерывание по окончании ввода-вывода является вторым по важности входом в MPX. Когда происходит такое прерывание, прерываемый рабочий запрос помещается в очередь AQ, где он и остается в качестве очередного готового к обработке запроса. Затем слово запроса ввода-вывода для передачи данных, об окончании которой получен сигнал, восстанавливается по значению соответствующего слова канала в списке обращений. Этот запрос ввода-вывода используется для увеличения счетчика буфера в поле запроса ввода-вывода, а также чтобы определить, был ли первоначальный запрос ЦП отложен до окончания передачи этих данных. Если это имеет место, то слово ЦП для отложенного рабочего запроса помещается в очередь AQ, делая его доступным для обработки. Кроме того, отложенный запрос ввода-вывода отличается тем, что запись, затребованная запросом ЦП, отсутствие которой вызвало прерывание обработки этого запроса, становится теперь доступной для него. С другой стороны, если запрос ввода-вывода, передача данных для которого только что закончилась, не нес ответственности за свой прерванный исходный запрос ЦП, тогда следующим действием должен быть опрос очереди IQ на канал, в котором передача данных уже закончилась для какого-либо запроса ввода-вывода, ожидавшего доступ к этому каналу. Раз эти действия осуществлены, то последним действием MPX является выбор очередного текущего обрабатываемого запроса из очереди AQ.

Более подробно эти действия изображены на рис. 35. Работа начинается в блоке 1, где прерванный запрос ЦП  $T$  пересекает границу ОС. При этом запрос операционной системы Z становится текущим обрабатываемым запросом, а  $T$  становится запросом, ожидающим



обработки. Кроме того, запрещаются прерывания ввода-вывода. Напомним, что, пытаясь найти вход в очередь AQ, запрос  $Z$  входит в оператор цикла с одновременным разрешением прерываний по окончании ввода-вывода. В таком случае, если случится прерывание по окончании

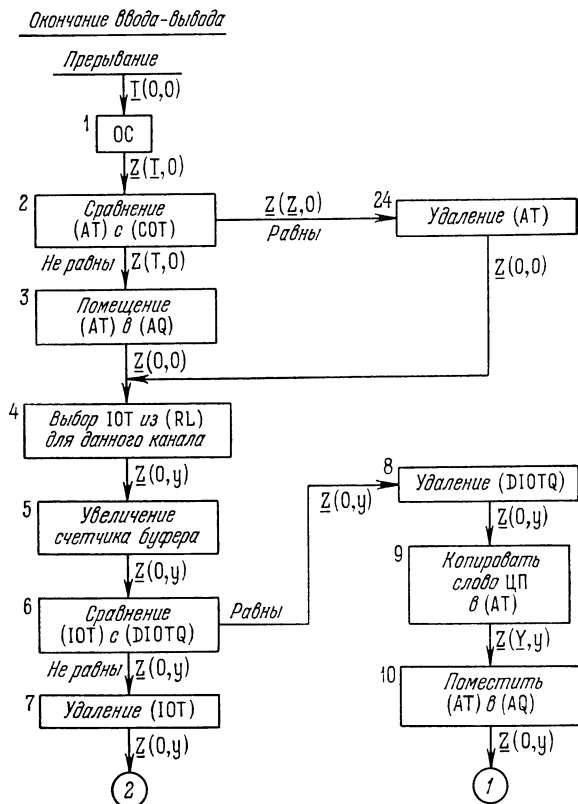


Рис. 35. Вход в МРХ по прерыванию окончания ввода-вывода.

ввода-вывода, произойдет прерывание обработки запроса  $Z$ . Блок 2 и служит как раз для того, чтобы распознать такую ситуацию. Это выполняется путем сравнения AT с COT. Если система была в режиме ожидания во время прерывания по окончании ввода-вывода, тогда запрос ЦП ( $T$ ), встречающийся в выражении ЦП на входе блока 2, на самом деле является запросом опе-

рационной системы  $Z$  и должен совпасть с СОР. При этом произойдет переход в блок 24, где АТ уничтожается.

С другой стороны, если ОС не была в режиме ожидания во время прерывания, то прерываемый запрос

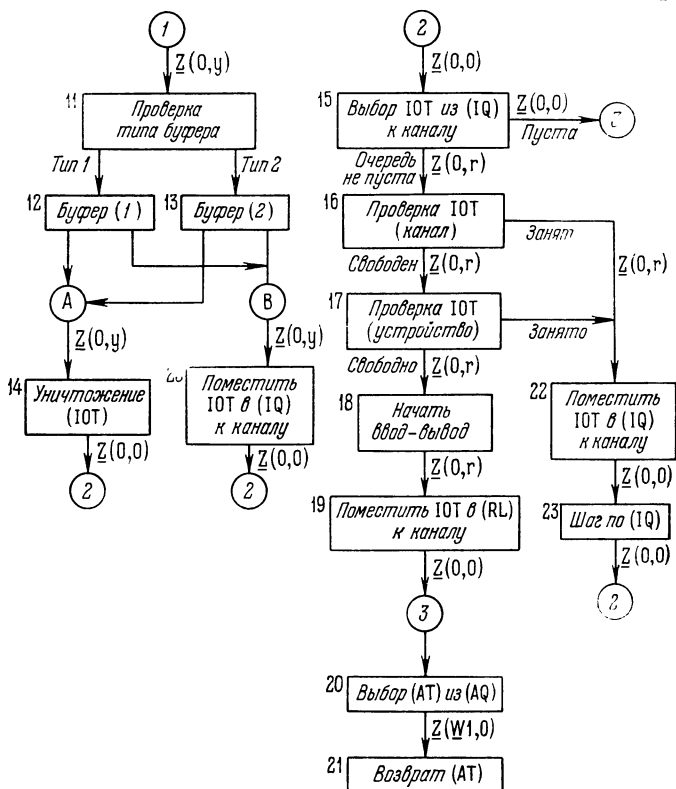


Рис. 35. Вход в МРХ по прерыванию окончания ввода-вывода (продолжение).

связан с рабочей программой, и поэтому сравнения не произойдет. В этом случае в блоке 3 готовый к обработке запрос, т. е. запрос рабочей программы, прерываемый по окончании ввода-вывода, помещается в очередь АQ и может быть выбран впоследствии в качестве СОР.

Тогда в любом из описанных случаев на входе в блок 4 имеется либо АТ, либо IOT. Поэтому следующим

действием является восстановление слова запроса ввода-вывода, связанного с прерыванием по окончании ввода-вывода. Оно должно быть найдено в списке RL для окончившего работать канала. Именно это и делается в блоке 4.

В блоке 5 значение счетчика буфера в поле запроса ввода-вывода увеличивается на 1, указывая тем самым на то, что для запроса ЦП, сформировавшего этот запрос ввода-вывода, доступен еще один блок буфера. Блок 5 не изменяет выражения ЦП.

Далее определяется, был ли какой-либо запрос рабочей программы прерван во время ожидания окончания обмена по IOT. Напомним, что если рабочий запрос ищет доступа к записи в буфере, который находится в состоянии С, его обработка прерывается. Кроме того, слово ввода-вывода, которое он сформировал, помещается в очередь DIOTQ. Поэтому, если слово ввода-вывода, представляющее окончившуюся передачу данных, связано с таким буфером, то сравнение IOT с очередью DIOTQ произойдет успешно. В этом случае произойдет переход в блок 8. Отметим, что работа блока 6 не влияет на выражение ЦП.

Предположим, что IOT удачно сравнился с очередью DIOTQ. Тогда в блоке 8 происходит удаление этого запроса из очереди DIOTQ, так как данный IOT идентичен ему и может быть теперь использован для того, чтобы представлять в этой очереди этот удаленный запрос ввода-вывода. В блоке 9 копия слова ЦП из поля запроса ввода-вывода, которое идентично исходному слову запроса ЦП (удаленное ранее как часть отложенного процесса, блок 16, рис. 34), копируется в ячейку AT поля ЦП. Это указывает на необходимость присутствия слова ЦП в каждом поле запроса ввода-вывода, так как монитору при работе требуется слово ЦП, а единственной информацией о нем является IOT. Этой цели служит поле запроса ввода-вывода, в котором помещена копия исходного слова ЦП. Итак, на выходе из блока 9 указано, что исходным запросом ЦП (обозначенным здесь Y) является теперь AT.

Так как исходный запрос ЦП теперь не находится в отложенном состоянии, он помещается в очередь запросов, ожидающих обработки AQ, откуда он может быть выбран в качестве COT. Это выполняется в блоке 10.

К этому моменту ИОТ представляет запрос ввода-вывода, который ранее вызвал прерывание выполнения исходного запроса ЦП. Поэтому необходимо так обработать этот запрос ввода-вывода, чтобы запись, которая требовалась исходному запросу, снова стала доступной для него (удаляя или помещая ее в зависимости от того, является ли буфер буфером ввода или буфером вывода). Чтобы выполнить это, необходимо обратиться к оператору буфера. В блоке 11 определяется тип буфера, связанного с данным ИОТ, и затем работает блок 12 или 13, в зависимости от результата работы предыдущего блока.

Так как вход в МРХ был произведен по окончании ввода-вывода, представляющего передачу данных, которая закончилась для буфера, связанного с этим ИОТ, то этот буфер уже не может находиться в состоянии С. С другой стороны, если коэффициент блока больше единицы, т. е. в блоке содержится более одной записи, тогда ясно, что буфер находится в состоянии А. В этом случае работает блок 14.

Однако, если коэффициент блока равен единице, т. е. в блоке всего одна запись, тогда буфер находится в состоянии В и управление передается в блок 25. В первом случае, так как ИОТ уже сделал свое дело, его можно удалить, оставив выражение ЦП без АТ или ИОТ, после чего передать управление в блок 15. Однако, если коэффициент блока равен единице, тогда при работе блока 25 ИОТ помещается в очередь на обмен в канале. При этом выражение ЦП остается либо без АТ, либо без ИОТ, а управление передается также в блок 15.

Здесь мы прервем описание работы МРХ. Во-первых, если прерванный запрос ЦП был связан с рабочей программой, он был помещен в очередь АQ, откуда его можно выбрать в обработку. Слово ввода-вывода, представляющее окончившуюся передачу данных, восстанавливается и используется для увеличения счетчика буфера в поле запроса ввода-вывода. Затем он используется для того, чтобы определить, был ли отложен исходный запрос ЦП из-за того, что буфер находился в состоянии С при последнем запросе ввода-вывода. Это выполняется при помощи сравнения ИОТ с очередью отложенных запросов ввода-вывода. Если сравнения не произошло, то либо запрос ЦП не отложен (в этом

случае он находится в очереди AQ), либо запрос в самом деле отложен, но это произошло благодаря тому, что какой-либо из других буферов, связанных с этим запросом, находился в состоянии С. Во втором случае следует ждать окончания ввода-вывода для этого буфера, чтобы можно было выйти из состояния ожидания. В любом случае неудачное сравнение IOT с членами очереди DIOTQ предполагает, что никаких действий по выводу исходного запроса ЦП из состояния ожидания предпринимать не надо. Следовательно, IOT, выполнивший все свои функции, уничтожается. Если же исходный запрос ЦП был все-таки отложен, ожидая завершения передачи данных, то копия слова ЦП переписывается из поля запроса ввода-вывода и помещается в очередь AQ, откуда исходный запрос ЦП может быть затем выбран в качестве СОТ. Затем IOT используется для выбора отложенного запроса ввода-вывода, начиная затем ввод-вывод, если это необходимо в данном состоянии буфера ввода-вывода. Во всех случаях управление передается в точку 2 (рис. 35).

В блоке 15 выбирается первая (в логическом смысле) запись из очереди заявок на канал, в котором обмен закончился (если блок 25 был последним работавшим блоком, то в этой очереди есть по крайней мере одна запись).

Затем управление передается в блок 16, а отсюда в блоки 17, 18, 19 или 22, где, как уже описывалось выше, делается попытка начать ввод-вывод, связанный с выбранным IOT. Если попытка оказалась удачной, тогда начинается работа по вводу-выводу (блок 18), а IOT помещается в список заявок к данному каналу (блок 19). Если же невозможно начать ввод-вывод по данному IOT, то он возвращается назад в очередь, и из этой очереди выбирается следующий запрос (блок 23), после чего управление передается в блок 15.

Этот цикл блок 23 — блок 15 повторяется до тех пор, пока либо не найдется такого IOT, для которого можно начать ввод-вывод, либо пока не будут просмотрены все члены очереди. В последнем случае осуществляется выход «пусто» из блока 16. После этого происходит переход в точку 3 (блок 20).

Теперь все операции, относящиеся к началу ввода-вывода в канале, закончившем работу, выполнены. Последнее, что делает МРХ в этом случае, это выбирает

очередной СОТ из очереди АQ (блок 20), делая его текущим обрабатываемым запросом. Затем, возвращаясь к нему в блоке 21, начинает его обработку. Этот же оператор разрешает прерывания по окончании ввода-вывода.

В том случае, если память может быть распределена, производится ее распределение. Область отведенной памяти запоминается в поле запроса, связанного со словом запроса, обрабатываемого в настоящий момент загрузчиком. Затем загружается необходимая программа, и по завершении процесса загрузки слово запроса только что загруженной программы помещается в АQ. Таким образом, программа становится одной из программ мультипрограммного режима.

## СОКРАЩЕНИЯ

AL — список задач, готовых к обработке.

AQ — очередь задач, готовых к обработке, очередь готов-  
гости.

AT — запрос, готовый к обработке, запрос в состоянии го-  
товности.

COMCNT — счетчик, указывающий число задач, обрабатывае-  
мых компилятором.

COML — список задач, ожидающих загрузки компилятором.

COP — текущая обрабатываемая программа.

COT — текущий обрабатываемый запрос.

CTDQ — очередь запросов на ввод перфокарт.

DIRL — список задач к директору.

DIRQ — очередь задач к директору.

DTRQ — очередь задач к планировщику.

DSPQ — очередь задач к диспетчеру.

DIOTQ — очередь отложенных запросов на ввод-вывод.

IOT — запрос на ввод-вывод.

IQ — очередь задач, ожидающих ввода-вывода

ID — идентификатор программы.

IL — список задач, ожидающих ввода-вывода

LDRQ — очередь задач к загрузчику.

MPX — мультипрограммный исполнитель, монитор.

OS — ОС — операционная система.

RCVQ — очередь задач к приемнику.

RL — список задач, ожидающих окончания ввода-вывода,  
список ссылок.

*Л. Дж. Козн*

**АНАЛИЗ И РАЗРАБОТКА  
ОПЕРАЦИОННЫХ СИСТЕМ**

(Серия: «Библиотечка программиста»)

М., 1975 г., 192 стр. с илл.

Редактор *Г. Я. Пирогова*

Технический редактор *В. Н. Кондакова*

Корректоры *О. А. Бутусова, А. Л. Ипатова*

Сдано в набор 27/VI 1975 г.

Подписано к печати 28/X 1975 г.

Бумага 84×108<sup>1</sup>/<sub>32</sub>. Физ. печ. л. 6.

Условн. печ. л. 10,08. Уч.-изд. л. 9,64.

Тираж 30 000 экз. Цена 69 коп.

Заказ № 3058.

Издательство «Наука»

Главная редакция

физико-математической литературы

117071, Москва, В-71,

Ленинский проспект, 15

Ордена Трудового Красного Знамени

Первая Образцовая типография

имени А. А. Жданова

Союзполиграфпрома

при Государственном комитете

Совета Министров СССР по делам издательств

полиграфии и книжной торговли.

Москва, М-54, Валовая, 28

Отпечатано во 2-ой типографии

издательства «Наука».

Москва, Шубинский пер., 10.



ИЗДАТЕЛЬСТВО «НАУКА»  
ГЛАВНАЯ РЕДАКЦИЯ  
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ  
117071, Москва, В-71, Ленинский проспект, 15

**В СЕРИИ «БИБЛИОТЕЧКА ПРОГРАММИСТА»  
ГОТОВЯТСЯ К ПЕЧАТИ:**

**Карпов В. Я., Алгоритмический язык ФОРТРАН**, монография, под ред. Н. Н. Говоруна, М., Главная редакция физико-математической литературы изд-ва «Наука», 10 л., 63 к., 20204.

В книге дано подробное описание алгоритмического языка ФОРТРАН. Язык ФОРТРАН пользуется все большей популярностью, особенно в связи с введением в эксплуатацию транслятора с ФОРТРАНа, предназначенного для одной из самых мощных отечественных ЭВМ—БЭСМ-6. Книга может служить учебным пособием для желающих изучить основы языка и справочным руководством для программистов, работающих на дубнинской версии языка, входящей в стандартное математическое обеспечение машины БЭСМ-6.

Материал иллюстрирован многочисленными примерами и сопровождается упражнениями.

**Салтыков А. И., Макаренко Г. И., Программирование на языке ФОРТРАН**, монография, под ред. Н. Н. Говоруна, М., Главная редакция физико-математической литературы изд-ва «Наука», 10 л., 63 к., 20204.

В книге излагаются основы программирования на алгоритмическом языке ФОРТРАН и автокоде MADLEN для БЭСМ-6. Изложение ведется применительно к конкретной версии языка ФОРТРАН, принятой в мониторной системе «Дубна». Приводятся необходимые сведения о машине БЭСМ-6 и ее математическом обеспечении. Специальная глава посвящена вопросам оптимизации программ.

Книга рассчитана на широкий круг программистов — вычислителей, использующих БЭСМ-6; она может также служить учебным пособием для студентов вузов и техникумов.

Книги В. Я. Карпова и А. И. Салтыкова, Г. И. Макаренко вместе дают достаточно полную информацию о ФОРТРАНе в версии «Дубна» применительно к БЭСМ-6.

*Предварительные заказы на печатающиеся книги принимаются без ограничения всеми магазинами Книготорга и Академкниги. При отказе в приеме заявок заказы можно направлять по адресу: 103050, Москва, К-50, ул. Медведова, 1, отдел «Книга — почтой» магазина № 8 Москниги.*

69 коп.