Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ
И РАДИОЭЛЕКТРОНИКИ

---

Кафедра информационных технологий автоматизированных систем

А.М.Севернёв

# ОПЕРАЦИОННЫЕ СИСТЕМЫ

**Методическое пособие для выполнения лабораторных работ
по курсу “Операционные системы”**

**для студентов специальности 53 01 02
“Автоматизированные системы обработки информации и управления”**



Минск 2002

УДК 681.3.066 (075.8)
ББК 32.973 я73
С 28

**Севернёв А.М.**

С 28 Операционные системы: Метод. пособие для выполнения лаб. работ по курсу “Операционные системы” для студентов спец. 53 01 02 – “Автоматизированные системы обработки информации и управления”/ А.М Севернёв. – Мн.: БГУИР, 2002. – 99 с.

ISBN 985 – 444 – 391 – 4.

Методическое пособие для выполнения лабораторных работ по курсу “Операционные системы” содержит восемь лабораторных работ и предназначено как для изучения основных механизмов операционных систем с помощью моделей, так и для приобретения практических навыков работы с конкретными ОС.

УДК 681.3.066 (075.8)
ББК 32.973 я73



ISBN 985 – 444 – 391 – 4

СОДЕРЖАНИЕ

# 1 КОМПЛЕКС МОДЕЛЕЙ УПРАВЛЕНИЯ ПАМЯТЬЮ

## 1.1 Цель работы

**1.1.1** Получить сведения об основных методах распределения памяти.

**1.1.2** Изучить влияние внутренних параметров конкретных методов управления памятью в мультипрограммных системах на выходные параметры ВС с помощью имитационных GPSS-моделей.

## 1.2 Управление памятью в мультипрограммных ОС

**1.2.1** Функции ОС по управлению памятью

**1.2.1.1** Основными функциями ОС по управлению памятью в мультипрограммной системе являются [4, 5, 8–11, 13]:

– отслеживание свободной и занятой памяти;

– выделение памяти процессам и освобождение памяти по завершении процессов;

– вытеснение кодов и данных процессов из оперативной памяти (ОП) на диск (полное или частичное), когда размеры основной памяти недостаточны для размещения в ней всех процессов, и возвращение их в ОП, когда в ней освобождается место;

– настройка адресов программы на конкретную область физической памяти.

**1.2.1.2** Помимо первоначального выделения памяти процессам при их создании ОС должна также заниматься динамическим распределением памяти, т.е. выполнять запросы приложений на выделение им дополнительной памяти во время выполнения. Еще одной важной задачей ОС является защита памяти, состоящая в том, чтобы не позволить выполняемому процессу записывать или читать данные из памяти, назначенной другому процессу.

**1.2.2** Типы адресов

**1.2.2.1** Для идентификации переменных и команд на разных этапах жизненного цикла программы используются (рисунок 1.1):

– *символьные имена*; их присваивает пользователь при написании программы на алгоритмическом языке или ассемблере.

– *виртуальные (математические или, иначе, логические) адреса*; их вырабатывает транслятор, переводящий программу на машинный язык. Поскольку во время трансляции в общем случае неизвестно, в какое место оперативной памяти будет загружена программа, то транслятор присваивает переменным и командам виртуальные (условные) адреса, обычно считая по умолчанию, что начальным адресом программы будет нулевой адрес;

– *физические адреса* – соответствуют номерам ячеек оперативной памяти,

где в действительности расположены или будут расположены переменные и команды.

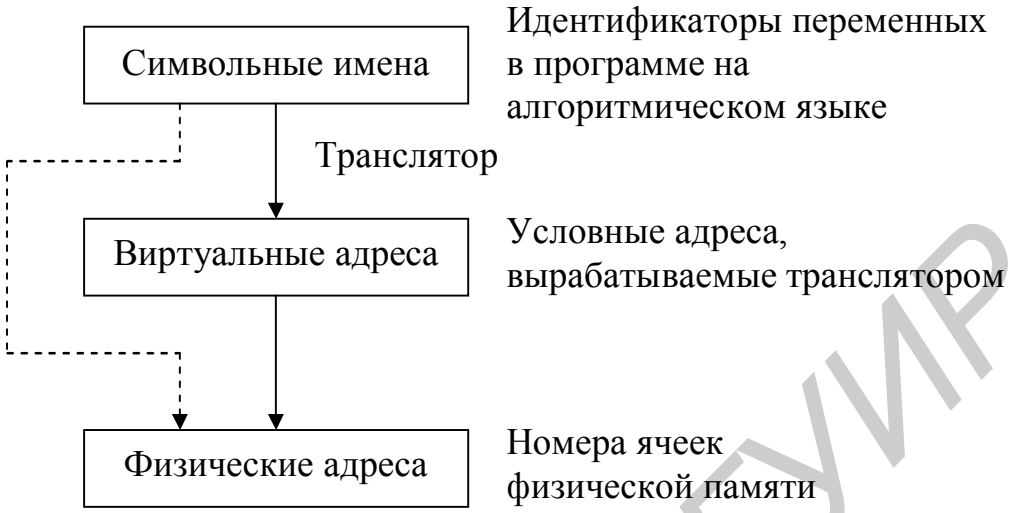



Рисунок 1.1 – Типы адресов

**1.2.2.2** Совокупность виртуальных адресов процесса называется *виртуальным адресным пространством*. Диапазон возможных адресов виртуального пространства у всех процессов является одним и тем же. Например, при использовании 32–разрядных виртуальных адресов этот диапазон задается границами $00000000_{16}$ и $FFFFFFFF_{16}$. Тем не менее каждый процесс имеет собственное виртуальное адресное пространство – транслятор присваивает виртуальные адреса переменным и кодам каждой программы независимо друг от друга. Причем совпадение виртуальных адресов переменных и команд различных процессов не приводит к конфликтам, так как в том случае, когда эти переменные одновременно присутствуют в памяти, ОС отображает их на разные физические адреса.

Необходимо различать *максимально возможное* виртуальное адресное пространство процесса и *назначенное (выделенное)* процессу виртуальное адресное пространство. В первом случае речь идет о максимальном размере виртуального адресного пространства, определяемом архитектурой компьютера, в частности разрядностью его схем адресации (32–битная, 64–битная и т.п.). Заметим, что максимальный размер виртуального адресного пространства , как правило, не совпадает с объемом физической памяти, имеющимся в компьютере (именно на этом факте и использовании внешней памяти основана виртуальная память). Назначенное же виртуальное адресное пространство представляет собой набор виртуальных адресов, действительно нужных процессу для работы. Эти адреса первоначально назначает программе транслятор на основании текста программы, когда создает кодовый сегмент, а также сегмент (сегменты) данных, с которыми программа работает. Затем при создании процесса ОС фиксирует назначенное виртуальное адресное пространство в своих системных таблицах.

**1.2.2.3** В разных ОС используются разные способы структуризации

виртуального адресного пространства:

– в виде непрерывной *линейной* последовательности виртуальных адресов; такую структуру адресного пространства называют также *плоской (flat),* при этом виртуальным адресом является единственное число *m*, называемое линейным виртуальным адресом;

– в виде частей, называемых *сегментами* (секциями, областями или другими терминами); в этом случае виртуальный адрес представляет собой *пару* чисел (*n, m*), где *n* определяет сегмент, а *m* – внутрисегментное смещение.

Существуют и более сложные способы структуризации виртуального адресного пространства, когда виртуальный адрес образуется тремя и даже более числами.

**1.2.2.4** Задачей ОС является отображение индивидуальных виртуальных адресных пространств всех одновременно выполняющихся процессов на общую физическую память. Существуют два принципиально различающихся подхода к преобразованию виртуальных адресов в физические:

– замена виртуальных адресов на физические выполняется один раз для каждого процесса во время начальной загрузки программы в ОП с помощью специальной системной программы – *перемещающего загрузчика*;

– программа загружается в память в неизменном виде в виртуальных адресах, выработанных транслятором (см. рисунок 1.1). При загрузке ОС фиксирует смещение S действительного расположения программного кода относительно виртуального адресного пространства, занося его на время выполнения программы в специальный регистр процессора. Во время выполнения программы при каждом обращении к ОП виртуальные адреса данной программы преобразуются в физические путем прибавления к ним смещения S.

Последний способ является более гибким: в то время как перемещающий загрузчик жестко привязывает программу к первоначально выделенному ей участку памяти, динамическое преобразование виртуальных адресов позволяет перемещать программный код процесса в течение всего времени его выполнения. Но использование перемещающего загрузчика более экономично, так как в этом случае преобразование каждого виртуального адреса происходит только один раз во время загрузки, а при динамическом преобразовании – при каждом обращении по данному адресу.

**1.2.3** Алгоритмы распределения памяти

**1.2.3.1** Все алгоритмы распределения памяти разделены на два класса (рисунок 1.2): алгоритмы, в которых используется перемещение сегментов процессов между ОП и диском, и алгоритмы, в которых внешняя память не привлекается.

**1.2.3.2** Простейший способ управления оперативной памятью состоит в том, что память разбивается на несколько областей фиксированной величины, называемых *разделами*. Такое разбиение может быть выполнено вручную оператором во время старта системы или во время ее установки, после чего границы разделов не изменяются.

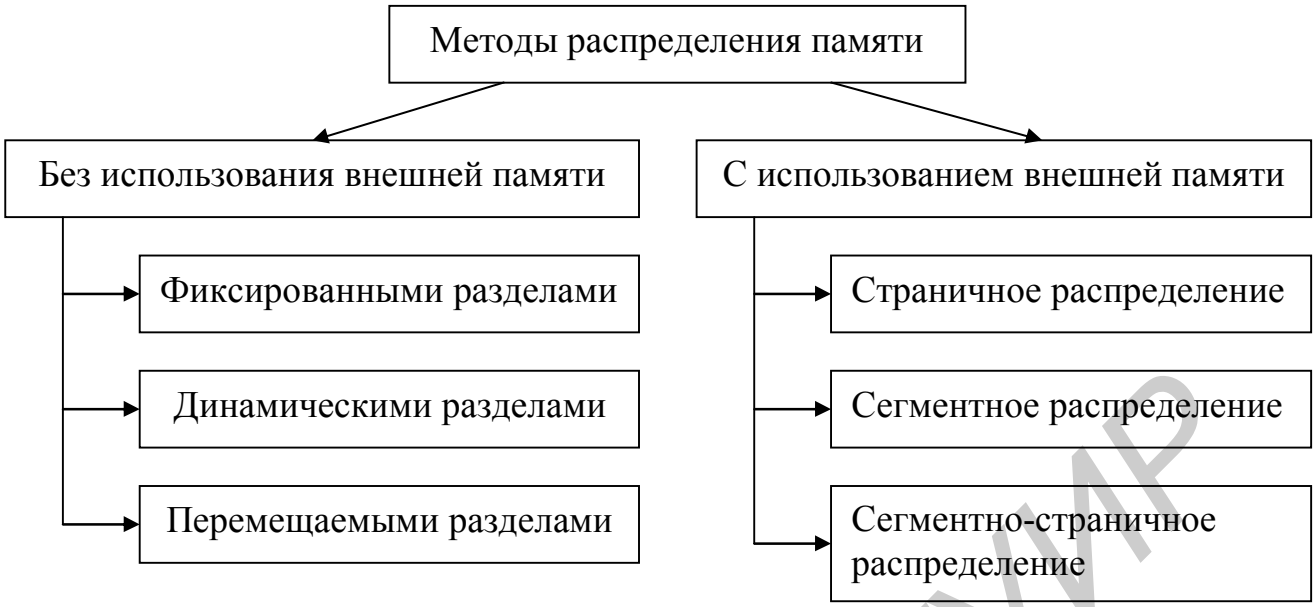



Рисунок 1.2 – Классификация методов распределения памяти

Очередной новый процесс, поступивший на выполнение, помещается либо в общую очередь (рисунок 1.3,а), либо в очередь к некоторому разделу (рисунок 1.3,б). Здесь ОС осуществляет загрузку программы в один из разделов, причем уже на этапе трансляции разработчик программы может задать раздел, в котором ее следует выполнять. Это позволяет сразу, без использования перемещающего загрузчика (см. подпункт 1.2.2.4), получить машинный код, настроенный на конкретную область памяти.

При очевидном преимуществе – простоте реализации данный метод управления памятью имеет существенный недостаток – жесткость. Так как в каждом разделе может выполняться только один процесс, то уровень мультипрограммирования заранее ограничен числом разделов (независимо от размера программы она будет занимать весь раздел). С другой стороны, разбиение памяти на разделы не позволяет выполнять процессы, программы которых не помещаются ни в один из разделов, но для которых было бы достаточно памяти нескольких разделов.

Такой способ управления памятью применялся в ранних мультипрограммных ОС. Однако и сейчас метод распределения памяти фиксированными разделами находит применение в системах реального времени, в основном благодаря небольшим затратам на реализацию. Детерминированность вычислительного процесса систем реального времени (т.е. заранее известен набор выполняемых задач, их требования к памяти, а иногда и моменты запуска) компенсирует недостаточную гибкость данного способа управления памятью.

**1.2.3.3** В случае распределения памяти *динамическими разделами* память машины не делится заранее на разделы, и сначала вся память, отводимая для приложений, свободна. Каждому вновь поступающему на выполнение приложению на этапе создания процесса выделяется вся необходимая ему память (если

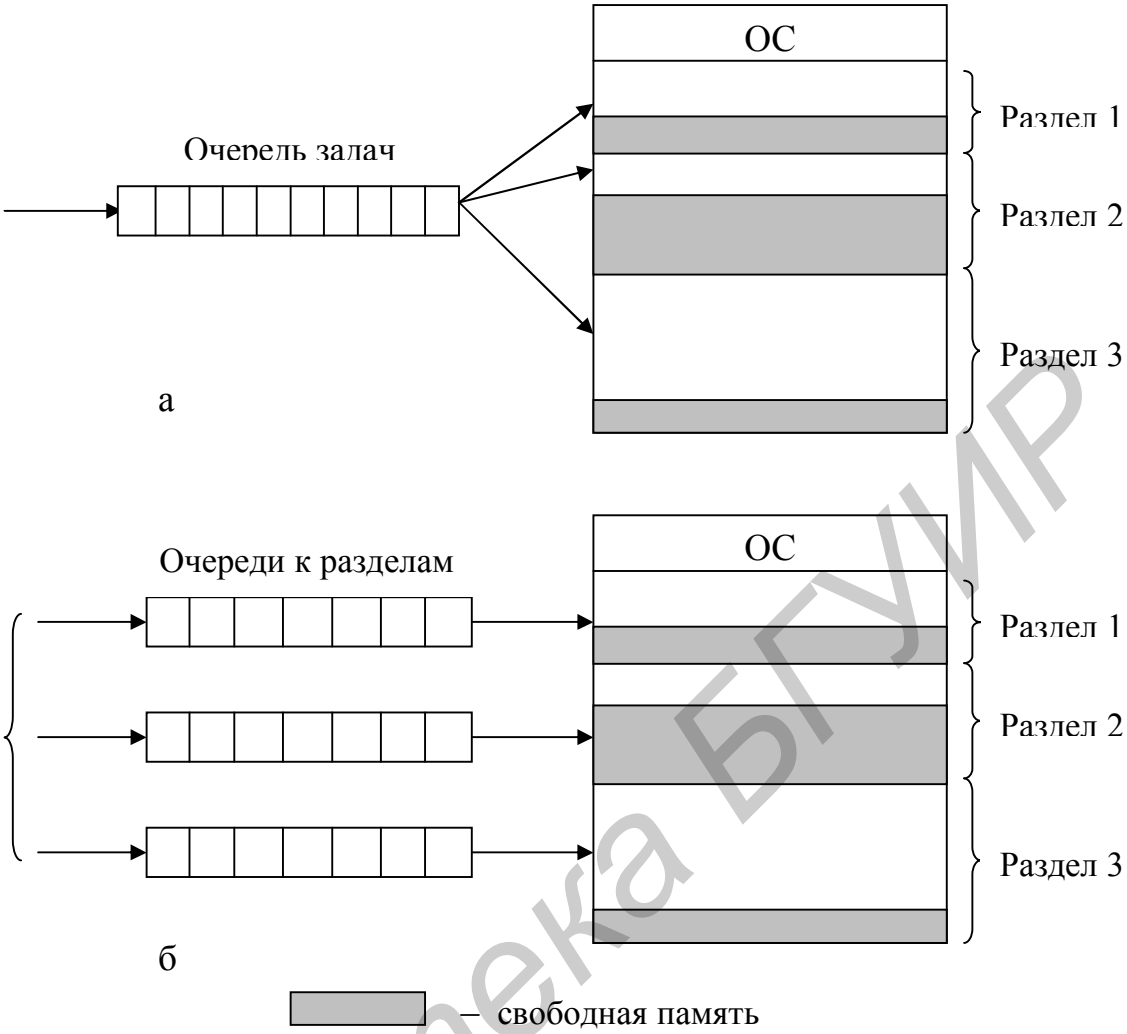



Рисунок 1.3 – Распределение памяти фиксированными разделами: с общей очередью (а), с отдельными очередями (б)

достаточный объем памяти отсутствует, то приложение не принимается на выполнение и процесс для него не создается). После завершения процесса память освобождается, и на это место может быть загружен другой процесс. На рисунке 1.4 показано распределение памяти динамическими разделами.

По сравнению с методом распределения памяти фиксированными разделами данный метод обладает большей гибкостью, но ему присущ очень серьезный недостаток – фрагментация памяти. *Фрагментация* – это наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов). Настолько маленького, что ни одна из вновь поступающих программ не может поместиться ни в одном из участков, хотя суммарный объем фрагментов может составить значительную величину, намного превышающую требуемый объем памяти. В принципе фрагментация памяти имеет место в любой вычислительной машине независимо от организации ее памяти. В мультипрограммных системах с фиксированными разделами фрагментация происходит либо потому, что задания пользователя в действительности не полностью занимают выделенные им разделы (хотя при этом считается, что процесс полностью занимает раздел), либо в случае,

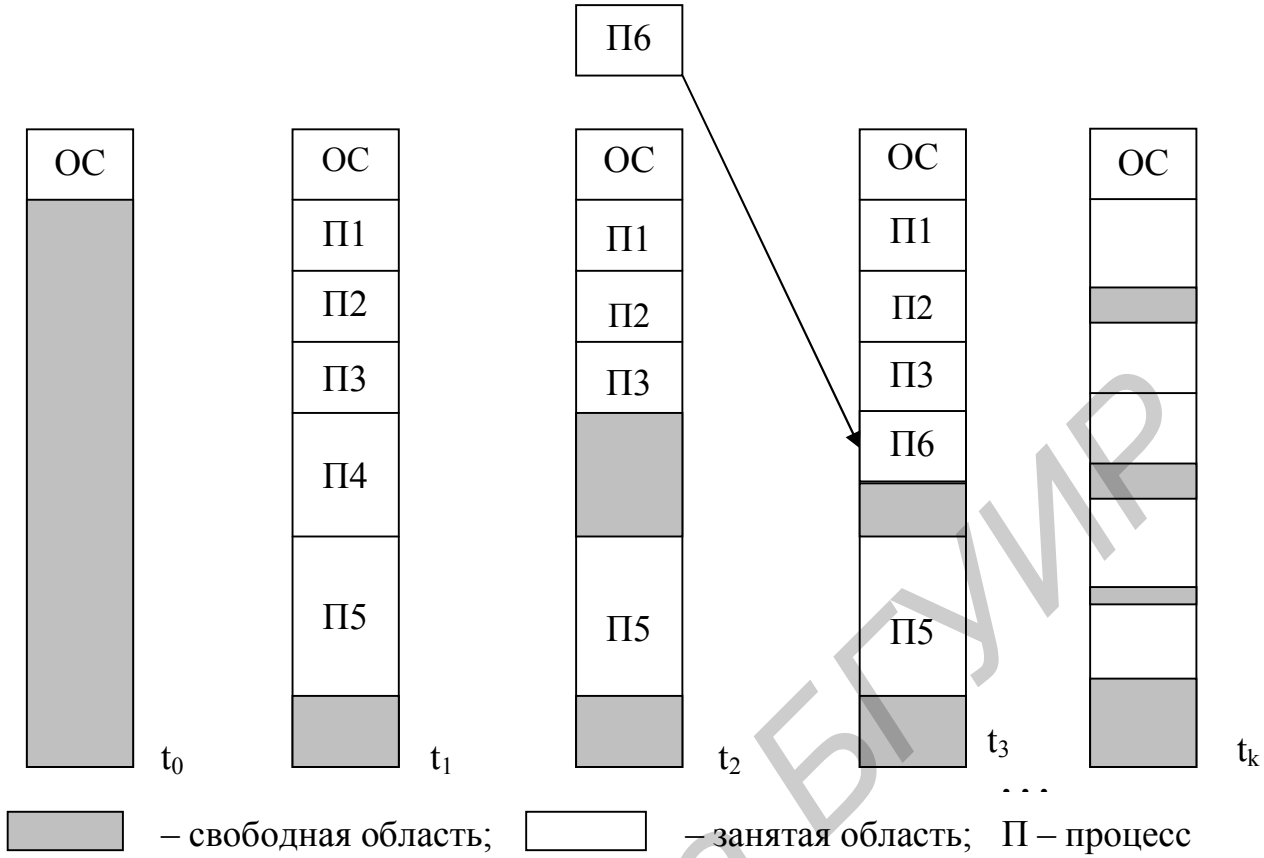



Рисунок 1.4 – Распределение памяти динамическими разделами

когда некоторый раздел остается незанятым из-за того, что он слишком мал для размещения ожидающего задания.

Распределение памяти динамическими разделами лежит в основе подсистем управления памятью многих мультипрограммных ОС 60–70-х годов прошлого века, в частности такой популярной ОС, как OS/360.

**1.2.3.4** Одним из методов борьбы с фрагментацией является перемещение всех занятых участков в сторону старших или младших адресов с тем, чтобы вся свободная память образовала единую свободную область. Эта процедура называется *сжатием (уплотнением) памяти*. Такой метод распределения памяти носит название распределения памяти *перемещаемыми разделами* (рисунок 1.5).

Сжатие может выполняться либо при каждом завершении процесса, либо только тогда, когда для вновь создаваемого процесса нет свободного раздела достаточного размера. В первом случае требуется меньше вычислительной работы при корректировке таблиц свободных и занятых областей, а во втором – реже выполняется процедура сжатия. Так как программы перемещаются по ОП в ходе своего выполнения, то в данном случае невозможно выполнить настройку адресов с помощью перемещающего загрузчика. Здесь более подходящим оказывается динамическое преобразование адресов (см. подпункт 1.2.2.4). Хотя процедура сжатия и приводит к более эффективному использованию памяти, она может потребовать значительного времени (накладных расходов), что часто перевешивает

преимущества данного метода (заметим, что во время уплотнения памяти система должна прекращать любые другие работы).

**1.2.3.5** Подмена (*виртуализация*) оперативной памяти дисковой (т.е. внешней) памятью позволяет повысить уровень мультипрограммирования – объем ОП компьютера в этом случае не столь жестко ограничивает количество одновременно выполняемых процессов, поскольку суммарный объем памяти, занимаемый образами этих процессов, может существенно превосходить имеющийся объем ОП (*образом* процесса называется совокупность его кодов и данных). *Виртуальным* называется ресурс, который пользователю или пользовательской программе представляется обладающим свойствами, которыми он в действительности не обладает. В данном случае в распоряжение прикладного программиста предоставляется виртуальная оперативная память, размер которой намного превосходит всю имеющуюся в системе реальную ОП. Пользователь пишет программу, а после ее трансляции система, используя виртуальные адреса, переводит ее в машинные коды так, как будто в распоряжении программы имеется однородная ОП большого объема. В действительности же все (или почти все) коды и данные, используемые программой, хранятся на дисках и только при необходимости загружаются в реальную ОП. Понятно, что работа такой “оперативной памяти” происходит значительно медленнее. Виртуализация ОП осуществляется операционной системой и аппаратурой процессора автоматически, без участия программиста, и никак не сказывается на логике работы приложений. Заметим, что концепция виртуальной памяти является далеко не новой. Впервые она была реализована в вычислительной машине Atlas, созданной в Манчестерском университете в Англии в 1960 г.

**1.2.3.6** Виртуализация памяти может быть осуществлена на основе двух различных подходов:

– *свопинг (swapping)* – образы процессов выгружаются на диск и возвращаются в ОП *целиком*;

– *виртуальная память (virtual memory)* – между оперативной памятью и диском перемещаются *части* (сегменты, страницы и т.п.) образов процессов.

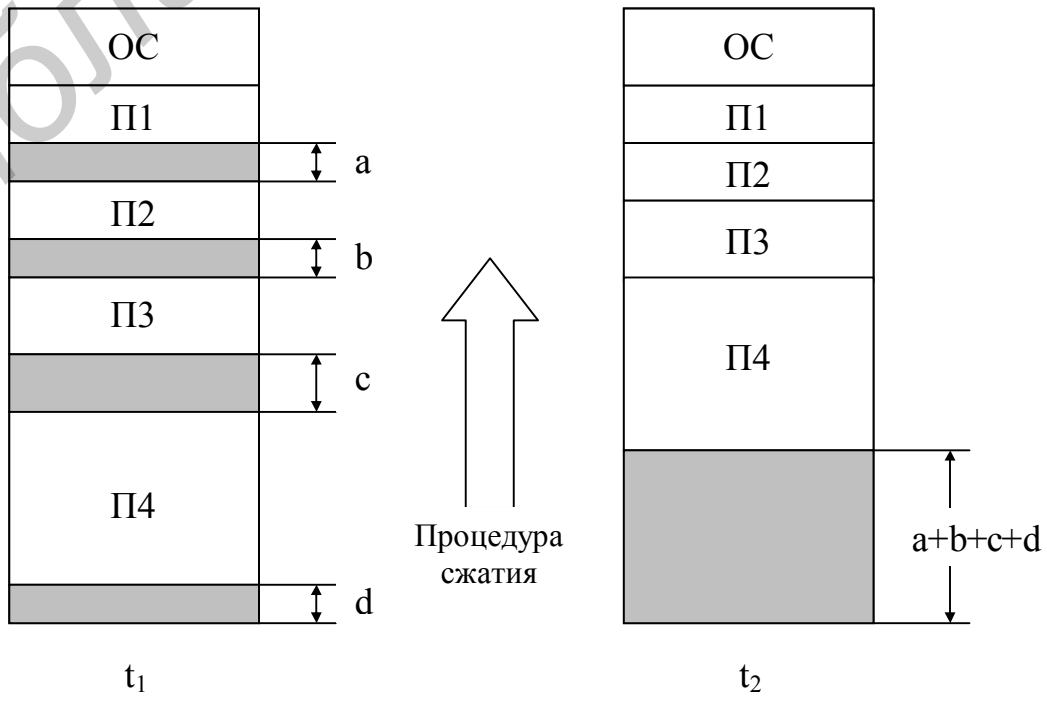



Рисунок 1.5 – Распределение памяти перемещаемыми разделами

**1.2.3.7** *Свопинг* представляет собой частный (крайний) случай виртуальной памяти, когда образы приостановленных процессов (ожидающих завершения ввода/вывода или освобождения ресурсов), а также образы готовых процессов могут быть временно, до следующего цикла активности, выгружены на диск. Несмотря на то, что коды и данные процесса отсутствуют в ОП, ОС "знает" о его существовании и в полной мере учитывает это при распределении процессорного времени и других системных ресурсов. К моменту, когда подходит очередь выполнения выгруженного процесса, его образ возвращается с диска в ОП. Если при этом обнаруживается, что свободного места в ОП не хватает, то на диск выгружается другой процесс.

Свопингу свойственна избыточность. Когда ОС решает активизировать процесс, то для его выполнения, как правило, не требуется загружать в ОП все его сегменты полностью – достаточно загрузить небольшую часть кодового сегмента с подлежащей выполнению инструкцией и часть сегмента данных, с которыми работает эта инструкция, а также отвести место под сегмент стека. Аналогично, при освобождении памяти для загрузки нового процесса очень часто вовсе не требуется выгружать другой процесс на диск целиком, достаточно вытеснить на диск только часть его образа. Перемещение избыточной информации замедляет работу системы, а также приводит к неэффективному использованию памяти. Кроме того, системы, поддерживающие свопинг, имеют еще один существенный недостаток: они не способны загрузить для выполнения процесс, виртуальное адресное пространство которого превышает имеющуюся в наличии свободную память.

Именно из-за указанных недостатков свопинг в качестве основного механизма управления памятью почти не используется в современных ОС. В некоторых современных ОС, например версиях UNIX, основанных на коде SVR4, механизм свопинга используется как дополнительный к виртуальной памяти, включающийся только при серьезных перегрузках системы.

**1.2.3.8** Ключевой проблемой виртуальной памяти, возникающей в результате многократного изменения местоположения в ОП образов процессов или их частей, является преобразование виртуальных адресов в физические. Решение этой проблемы, в свою очередь, зависит от того, какой способ структуризации виртуального адресного пространства принят в данной системе управления памятью. В настоящее время все множество реализаций виртуальной памяти может быть представлено тремя классами:

– *страничная виртуальная память* – организует перемещение данных между памятью и диском страницами, т.е. частями виртуального адресного пространства, фиксированного и сравнительно небольшого размера;

– *сегментная виртуальная память* – предусматривает перемещение данных сегментами, т.е. частями виртуального адресного пространства произвольного размера, полученными с учетом смыслового значения данных;

– *сегментно–страничная виртуальная память* – использует двухуровневое деление: виртуальное адресное пространство делится на сегменты, а затем сегменты делятся на страницы. Единицей перемещения данных здесь является страница. Этот способ управления памятью объединяет в себе элементы обоих предыдущих подходов.

Для вре́менного хранения сегментов и страниц на диске отводится либо специальная область, либо специальный файл, которые во многих ОС по традиции продолжают называть областью или файлом свопинга, хотя перемещение информации между ОП и диском осуществляется уже не в форме полного замещения одного процесса другим, а частями. Другое популярное название этой области – страничный файл (page file, или paging file). Текущий размер страничного файла является важным параметром, оказывающим влияние на возможности ОС: чем больше страничный файл, тем больше приложений может одновременно выполнять ОС (при фиксированном объеме ОП). Однако необходимо помнить, что увеличение числа одновременно работающих приложений за счет увеличения размера страничного файла замедляет их работу, так как значительная часть времени при этом тратится на перекачку кодов и данных из ОП на диск и обратно.

**1.2.3.9** В данной работе ограничимся рассмотрением лишь страничной виртуальной памяти, модель которой будет исследоваться в практической части работы. В этом случае адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые *виртуальными страницами (virtual pages)*. В общем случае размер виртуального адресного пространства процесса не кратен размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью.

Вся ОП машины также делится на части такого же размера, называемые *физическими страницами* (или блоками, или кадрами). Размер страницы выбирается равным степени двойки – 512, 1024, 4096 байт и т.д., что позволяет упростить механизм преобразования адресов.

При создании процесса ОС загружает в ОП несколько его виртуальных страниц (начальные страницы кодового сегмента и сегмента данных). Копия всего виртуального адресного пространства процесса находится на диске. Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах. Для каждого процесса ОС создает *таблицу страниц* – информационную структуру, содержащую записи обо всех виртуальных страницах процесса. Запись таблицы, называемая *дескриптором страницы*, включает следующую информацию:

– *номер физической страницы*, в которую загружена данная виртуальная страница;

– *признак присутствия*, устанавливаемый в единицу, если виртуальная страница находится в ОП;

– *признак модификации* страницы, который устанавливается в единицу всякий раз, когда производится запись по адресу, относящемуся к данной странице;

– *признак обращения* к странице, называемый также *битом доступа*, который устанавливается в единицу при каждом обращении по адресу, относящемуся к данной странице.

Информация из таблиц страниц используется для решения вопроса о необходимости перемещения той или иной страницы между памятью и диском, а также для преобразования виртуального адреса в физический. Сами таблицы страниц, так же как и описываемые ими страницы, размещаются в ОП. Адрес таблицы страниц (АТ) включается в контекст соответствующего процесса. При

активизации очередного процесса ОС загружает адрес его таблицы страниц в специальный регистр процессора.

При каждом обращении к памяти выполняется поиск номера виртуальной страницы, содержащей требуемый адрес, затем по этому номеру определяется нужный элемент таблицы страниц и из него извлекается описывающая страницу информация. Далее анализируется признак присутствия, и, если данная виртуальная страница находится в ОП, выполняется преобразование виртуального адреса в физический, т.е. виртуальный адрес заменяется указанным в записи таблицы физическим адресом (рисунок 1.6). Если же нужная виртуальная страница в данный момент выгружена на диск, то происходит так называемое *страничное прерывание*: выполняющийся процесс переводится в состояние ожидания, и активизируется другой процесс из очереди процессов, находящихся в состоянии готовности. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу (для этого ОС должна помнить положение вытесненной страницы в страничном файле диска) и пытается загрузить ее в ОП. Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно; если же свободных страниц нет, то на основании принятой в данной системе стратегии замещения страниц решается вопрос о том, какую страницу следует выгрузить из ОП.

Типичная машинная инструкция требует трех, четырех обращений к памяти (выборка команды, извлечение операндов, запись результата). И при каждом обращении происходит либо преобразование виртуального адреса в физический, либо обработка страничного прерывания. Время выполнения этих операций в значительной степени влияет на общую производительность вычислительной системы. Поэтому именно для уменьшения времени преобразования адресов во всех процессорах предусмотрен аппаратный механизм получения физического адреса по виртуальному.

Другим важным фактором, влияющим на производительность системы, является частота страничных прерываний, на которую, в свою очередь, влияют размер страницы и принятые в данной системе правила выбора страниц для выгрузки и загрузки. Например, на диск может выталкиваться страница, к которой в будущем (начиная с данного момента) дольше всего не будет обращений, или критерием выбора страницы на выгрузку является число обращений к ней за последний период времени, или при возникновении страничного прерывания может производиться так называемая *упреждающая загрузка* (загружается нужная виртуальная страница вместе с несколькими прилегающими к ней страницами).

**1.2.4** Исходные данные для моделей распределения памяти

**1.2.4.1** В данной работе в рамках методов распределения памяти (см. рисунок 1.2) без использования внешней памяти рассматриваются два алгоритма:

– распределение памяти фиксированными разделами (с отдельными очередями – модель stor11.gps и с общей очередью – модель stor11m.gps);

– распределение памяти перемещаемыми разделами (модель stor12.gps).

В рамках же методов распределения памяти с использованием внешней памяти рассматриваются следующие алгоритмы:

– алгоритм свопинга (модель stor13.gps);

– страничное распределение (страничная виртуальная память) (модель stor13.gps).

**1.2.4.2** Для всех моделей управления памятью (см. подпункт 1.2.4.1) приняты следующие допущения:

– объем оперативной памяти составляет 4 Мбайт (в моделях он задается в килобайтах, т.е. 4096 Кбайт);

– память, отводимая под ОС, не рассматривается, т.е. в моделях не учитывается;

– реализован вытесняющий алгоритм планирования процессов, основанный на квантовании, а именно режим разделения времени с циклическим планированием (RR) и квантом времени q (более подробно дисциплина планирования RR рассмотрена в разделе 6 настоящего пособия); накладные расходы системы, связанные с

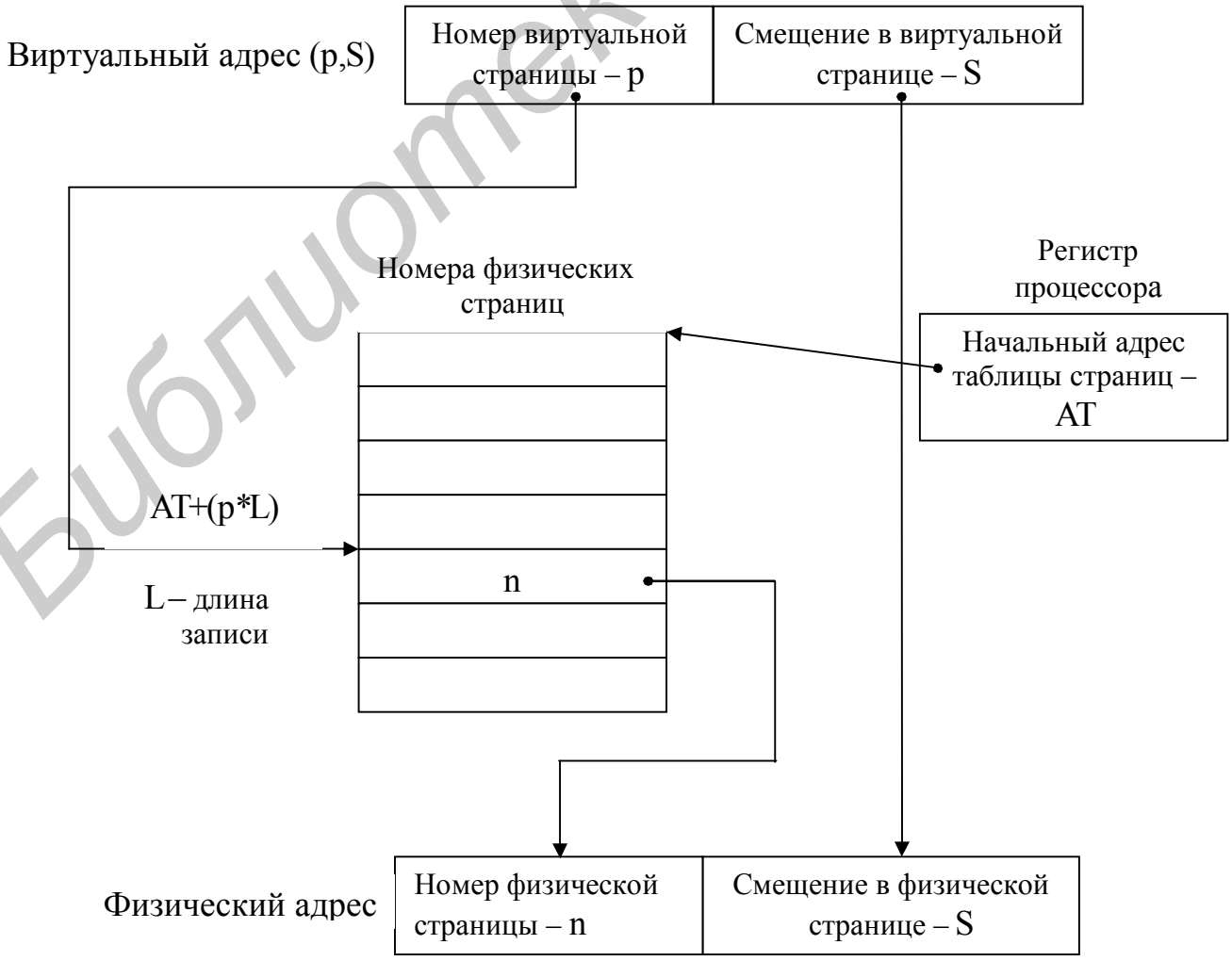



Рисунок 1.6 – Преобразование виртуального адреса в физический при страничной организации памяти

переключением процессора с одного процесса на другой, не учитываются;

– время обработки процесса на центральном процессоре и время нахождения процесса в состоянии ожидания из–за операции ввода/вывода подчиняются экспоненциальному закону распределения с математическими ожиданиями соответственно $\bar{t}_{CPU}$ и $\bar{t}_{I/O}$;

– поток задач в систему является простейшим;

– объем требуемой для процесса памяти является равномерно распределенной на интервале [16;128]x4 Кбайт случайной величиной и разыгрывается при каждом поступлении задачи в систему;

– на выделенном данному процессу кванте времени q в любой момент времени может произойти прерывание ввода/вывода (не связанного с загрузкой/выгрузкой страниц) с вероятностью $\mathbf{P}_{I/O}$=0,5 (в моделях это задается в поле операнда соответствующего оператора в виде значения, равного 500, т.е. в долях от тысячи).

**1.2.4.3** Для моделей управления памятью с фиксированными разделами (модели stor11.gps и stor11m.gps) вся ОП разбивается на три раздела разного размера, причем в случае отдельных очередей (модель stor11m.gps) их (очередей) тоже будет три.

**1.2.4.4** Для модели распределения памяти с перемещаемыми разделами (модель stor12.gps) процедура сжатия памяти выполняется при каждом завершении процесса, причем время ее выполнения $t_{сж}$ фиксировано. Процессор при выполнении системной программы уплотнения памяти не переключается на выполнение пользовательских процессов, т.е. ей выделяется подряд в общем случае несколько квантов q процессорного времени.

**1.2.4.5** Для всех моделей с использованием внешней памяти (модели stor13.gps и stor14.gps) начальный (максимальный) объем страничного файла равен 6 Мбайт (6144 Кбайт). Время загрузки/выгрузки всего образа процесса или его части определяется как объем перемещаемой части процесса в страницах, умноженный на 31 мкс.

**1.2.4.6** Для модели страничной виртуальной памяти оперативная память и страничный файл представляются как единая виртуальная память объемом 10 Мбайт (10240 Кбайт). При изменении объема страничного файла необходимо учитывать его нижнюю границу (2 Мбайт) и шаг изменения, кратный размеру страницы (4 Кбайт). Страничные прерывания возникают в случайные моменты времени, причем используется упреждающая загрузка (см. подпункт 1.2.3.9) и сразу загружается от трех до шести страниц.

## 1.3 Порядок выполнения работы

**1.3.1** Получить у преподавателя номер варианта.

**1.3.2** Скопировать в свой рабочий каталог D:\OS из каталога D:\OS\LABOS\LABOS1 файлы stor11.gps, stor11m.gps, stor12.gps, stor13.gps,

stor14.gps, lab11.bat, lab11m.bat, lab12.bat, lab13.bat, lab14.bat и startup.gps. Предварительно в рабочем каталоге должны быть размещены следующие обязательные для работы файлы: gpsspc.exe, gpssrept.exe, settings.gps и position.gps (из каталога D:\OS\GPSS).

**1.3.3** Изменить содержание поля операнда оператора (из файла stor11.gps)

VARIANT VARIABLE номер_варианта

в соответствии со своим вариантом.

**1.3.4** Получить результаты моделирования с помощью имитационной модели stor11.gps, запустив соответствующий batch-файл lab11.bat с учётом содержимого файла startup.gps (результаты смотри в файле stor11.txt).

**1.3.5** Запустить модель распределения памяти фиксированными разделами с общей очередью (модель stor11m.gps) с помощью файла lab11m.bat, предварительно задав полученный ранее номер своего варианта (аналогично пункту 1.3.3) и внеся соответствующие изменения в файл startup.gps (результаты см. в файле stor11m.txt).

**1.3.6** Сравнить между собой выходные результаты моделей распределения памяти фиксированными разделами с общей очередью и отдельными очередями и сделать выводы.

**1.3.7** Получить результаты моделирования распределения памяти перемещаемыми разделами (модель stor12.gps), предварительно задав свой номер варианта (по аналогии с пунктом 1.3.3) и запустив соответствующий batch-файл lab12.bat с учётом модифицированного содержимого файла startup.gps (результаты смотри в файле stor12.txt). Оценить накладные расходы системы по сжатию ОП в процентах от общего коэффициента загрузки процессора. Сравнить уровень мультипрограммирования (величина UR_MP в выходной статистике) в данной модели памяти с уровнем мультипрограммирования в системе с фиксированными разделами. Сделать выводы.

**1.3.8** Задать номер своего варианта в модели со свопингом (модель stor13.gps) аналогично пункту 1.3.3.

**1.3.9** Получить результаты моделирования алгоритма свопинга (см. файл stor13.txt), предварительно модифицировав файл startup.gps и запустив файл lab13.bat. Сравнить уровни мультипрограммирования данной системы с аналогичным показателем в системе с перемещаемыми разделами. Аналогичное сравнение провести для среднего времени решения задач (величина T_RESH в выходной статистике). Сделать выводы.

**1.3.10** Изменяя в файле stor13.gps вероятность $P_{I/O}$ перехода в состояние ожидания из–за операций ввода/вывода через изменение поля операнда

PIO VARIABLE вероятность_$P_{I/O}$_в_долях_от_тысячи

и получая результаты моделирования (см. пункт 1.3.9), построить график зависимости среднего уровня мультипрограммирования от интенсивности ввода/вывода, выраженной через вероятность $P_{I/O}$.

**1.3.11** Восстановить в модели stor13.gps начальное значение вероятности $P_{I/O}$ (см. подпункт 1.2.4.2 и пункт 1.3.10).

**1.3.12** Изменяя в файле stor13.gps объем страничного файла (см. подпункты 1.2.4.5, 1.2.4.6) через изменение поля операнда

SWP STORAGE объем _страничного_файла_в_килобайтах

и получая результаты моделирования (см. пункт 1.3.9), построить график зависимости среднего уровня мультипрограммирования от размера страничного файла.

**1.3.13** Задать номер своего варианта в модели страничного распределения памяти (модель stor14.gps) аналогично пункту 1.3.3.

**1.3.14** Получить результаты моделирования страничного распределения (см. файл stor14.txt), предварительно модифицировав файл startup.gps и запустив файл lab14.bat. Сравнить выходные результаты данной модели и модели со свопингом (см. пункт 1.3.9). По результатам сравнения сделать выводы.

**1.3.15** Изменяя в файле stor14.gps объем страничного файла (см. подпункты 1.2.4.5, 1.2.4.6) через изменение поля операнда

RAM_SWP STORAGE объем _виртуальной _памяти_в_килобайтах

и получая результаты моделирования (см. пункт 1.3.14), построить график зависимости среднего уровня мультипрограммирования от размера страничного файла.

**1.3.16** Восстановить в модели страничного распределения (файл stor14.gps) начальный размер виртуальной страничной памяти (см. подпункт 1.2.4.6 и пункт 1.3.15).

**1.3.17** Изменяя в файле stor14.gps вероятность $P_{I/O}$ перехода в состояние ожидания из–за операций ввода/вывода через изменение поля операнда

PIO VARIABLE вероятность_$P_{I/O}$_в_долях_от_тысячи

и получая результаты моделирования (см. пункт 1.3.14), построить графики зависимости среднего уровня мультипрограммирования и среднего времени решения задач от интенсивности ввода/вывода, выраженной через вероятность $P_{I/O}$.

**1.3.18** Оформить отчет и сформулировать общие выводы по работе (в виде текстового файла).

## 1.4 Контрольные вопросы

**1.4.1** Перечислите функции мультипрограммной ОС по управлению памятью.

**1.4.2** Какие типы адресов вы знаете?

**1.4.3** Какие подходы к преобразованию виртуальных адресов в физические вы знаете?

**1.4.4** Чем ограничивается максимальный размер физической памяти, которую можно установить в компьютере определенной модели?

**1.4.5** Что такое образ процесса?

**1.4.6** Перечислите известные методы распределения памяти.

**1.4.7** Чем определяется уровень мультипрограммирования в системах с распределением памяти фиксированными разделами?

**1.4.8** Что такое фрагментация памяти?

**1.4.9** Какое из этих двух утверждений верно: все виртуальные адреса заменяются на физические во время загрузки программы в ОП (утверждение А) или виртуальные адреса заменяются на физические во время выполнения программы в момент обращения по данному виртуальному адресу (утверждение В)?

**1.4.10** Распределение памяти перемещаемыми разделами основано на применении процедуры сжатия. Имеет ли смысл использовать данную процедуру при страничном распределении? А при сегментном?

**1.4.11** Что такое виртуальная память? Какой из следующих методов распределения памяти может рассматриваться как частный случай виртуальной памяти:

– распределение фиксированными разделами (ответ А);
– распределение динамическими разделами (ответ В);
– страничное распределение (ответ С);
– сегментное распределение (ответ D);
– сегментно–страничное распределение (ответ Е)?

**1.4.12** Поясните разные значения термина “свопинг”?

**1.4.13** Как величина файла подкачки влияет на производительность системы?

**1.4.14** На что влияет размер страницы? Каковы преимущества и недостатки большого размера страницы?

**1.4.15** Что такое страничное прерывание и какую последовательность действий в системе оно порождает?

# 2 ПРОГРАММНЫЕ ПРЕРЫВАНИЯ

## 2.1 Цель работы

**2.1.1** Получить представление об общей структуре компьютера.

**2.1.2** Ознакомиться с системой прерываний и некоторыми функциями API операционной системы MS–DOS.

**2.1.3** Получить начальные навыки в использовании программных прерываний DOS/BIOS из программы на языке высокого уровня.

## 2.2 Краткая характеристика компьютера

**2.2.1** Общее устройство компьютера

Типовая структурная схема компьютера фон-неймановской архитектуры представлена на рисунке 2.1 [2,14,15].

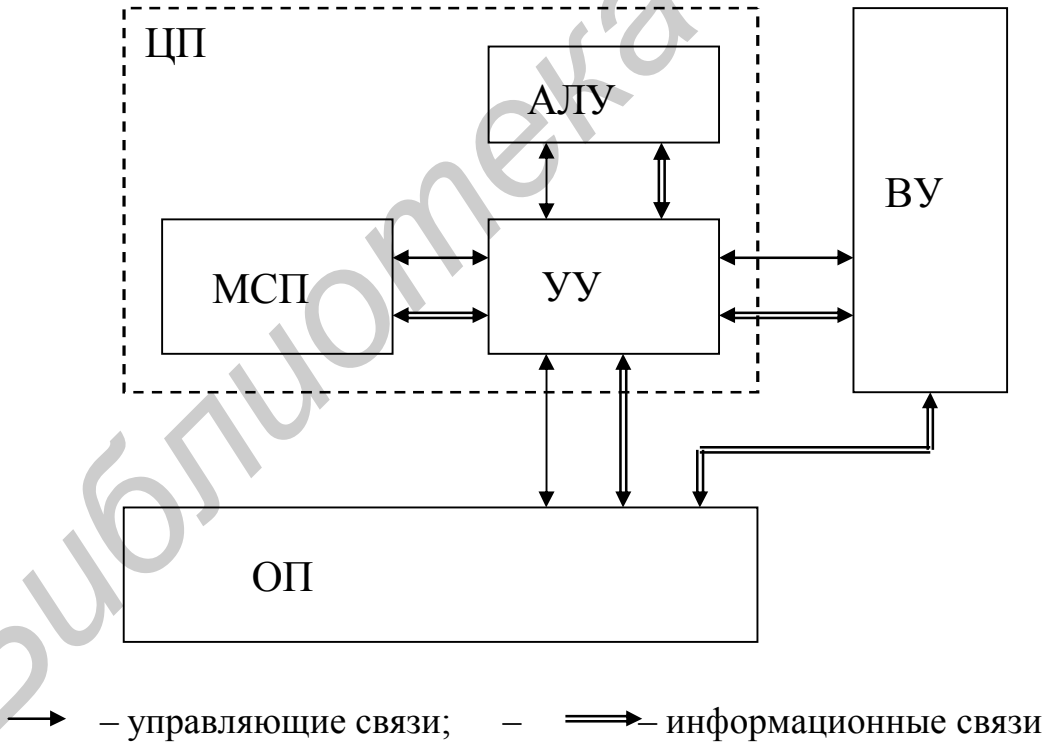



Рисунок 2.1 – Типовая структурная схема компьютера

*Арифметико–логическое устройство* (АЛУ, или ALU – Arithmetic and Logic Unit), *математический сопроцессор* (МСП, или FPU – Floating Point Unit) и

*устройство управления* (УУ, или CU – Control Unit) составляют *центральный процессор* (ЦП, или CPU – Central Processing Unit или, иначе, CP – Central Processor). *Оперативная память* (основная память, первичная память, физическая память, оперативное запоминающее устройство или запоминающее устройство с произвольной выборкой; англоязычные термины – read/write memory, random access memory, main memory, main stor, main storage, primary memory) (ОП) служит основным хранилищем информации, подлежащей обработке, т.е. в ней хранятся машинные программы, порождающие последовательности операций, и данные, участвующие в этих операциях в качестве операндов. УУ обеспечивает управление перемещением информации между ОП, АЛУ и *внешними устройствами* (устройствами ввода/вывода) (ВУ). Внешние устройства (дисплей, клавиатура, мышь, принтер, НЖМД, НГМД, CD–ROM, плоттер, джойстик и др.) обмениваются информацией с ЦП либо через *порты ввода/вывода* (в персональных компьютерах), либо через *каналы ввода/вывода* (в больших компьютерах типа мэйнфреймов). АЛУ в общем случае формирует функции двух входных переменных и порождает одну выходную переменную; эти функции обычно состоят из простых арифметических операций (сложение, вычитание, умножение и деление), простых логических операций (И, ИЛИ, НЕ, НЕ–И, НЕ–ИЛИ, исключающее ИЛИ, сложение по модулю 2, эквивалентность и др.) и операций сдвига. МСП предназначен для аппаратной поддержки математических операций над вещественными числами (числами с плавающей точкой) в случае, если такая поддержка отсутствует в основном процессоре. Современные процессоры (микропроцессоры) сами умеют выполнять операции над вещественными числами, поэтому для них сопроцессоры не требуются (они "встроены" в основной процессор).

**2.2.2** IBM PC–совместимые компьютеры

Большинство (более 90%) современных персональных компьютеров является IBM PC–совместимыми компьютерами. Эти компьютеры называются IBM PC–совместимыми, поскольку они совместимы (программно, а в значительной степени и аппаратно) с компьютером IBM PC, разработанным в 1981 г. крупнейшей в то время компьютерной фирмой IBM.

Важнейшую роль в развитии IBM PC–совместимых компьютеров сыграл заложенный в них фирмой IBM *принцип открытой архитектуры*, обеспечивающий сборку компьютера из независимо изготовленных частей аналогично детскому конструктору. В настоящее время термин "IBM PC " обычно используется в смысле "IBM PC–совместимый компьютер", а не как название компьютера, произведенного самой фирмой IBM.

**2.2.2** Архитектура микропроцессора Intel-8086/88

**2.2.2.1** Исходный вариант компьютера IBM PC и модель IBM PC XT используют микропроцессор (МП) Intel-8086/88 (приборы 8086 и 8088 различаются только разрядностью шины данных). Важность этого микропроцессора заключается в том, что его функции наследованы всеми моделями процессоров x86 и Pentium [13]. Более того, все процессоры популярного семейства 32–разрядных процессоров фирмы Intel – 80386, 80486, Pentium, Pentium Pro, Pentium II, Celeron, Pentium III и выше – для совместимости с программным обеспечением, разработанным для

предшествующих моделей процессоров Intel (главным образом, модели 8086), предусматривают так называемый *реальный режим* (real mode), хотя основным режимом работы этих процессоров является *защищенный режим* (protected mode). В реальном режиме (или, иначе, в режиме виртуального процессора 8086, если речь идет о режимах работы 32–разрядного процессора) процессор выполняет 16–разрядные инструкции и адресует 1 Мбайт памяти. Собственно защищенный режим характеризуется 4 Гбайт–адресным пространством ($2^{32}$), а также поддержкой многопрограммной работы. Перечисленные 32–разрядные процессоры имеют почти идентичные средства поддержки операционной системы, поэтому далее в тексте для их обозначения используется обобщенный термин “процессоры Pentium”.

**2.2.2.2** В состав прибора 8086/88 входят:

– микропрограммное устройство управления;

– арифметико–логическое устройство;

– программно–доступные регистры;

– регистр–указатель команд IP (Instruction Pointer) и другие внутренние регистры;

– блок формирования адресов;

– блок очереди команд;

– средства поддержки прямого доступа к памяти.

**2.2.2.3** В организации вычислительного процесса важную роль играют регистры процессора. Состав *программно–доступных регистров* МП 8086/8088 (а нас будут интересовать именно эти регистры) представлен на рисунке 2.2. По функциональному назначению они делятся на:

– регистры общего назначения (РОН);

– сегментные регистры;

– регистр флагов.

**2.2.2.4** РОНы в принципе допускают произвольное использование в программах, однако обычно они имеют вполне определенное назначение, соответствующее их названию. Это связано с тем, что некоторые команды задействуют те или иные регистры без их явного указания; кроме того, используются стандартные соглашения для связи программных модулей. В РОНах могут храниться данные или указатели адресов.

В состав РОН входят четыре 16–разрядных *информационных регистра* AX, BX, CX и DX, допускающие независимую адресацию старших (H) и младших (L) байтов. Информационные регистры стандартно выполняют следующие функции:

– *регистр-аккумулятор* AX используется для временного хранения данных и промежуточных результатов;

– *базовый регистр* BX служит для хранения указателя адреса области памяти, который участвует в формировании смещения при определенных режимах адресации;

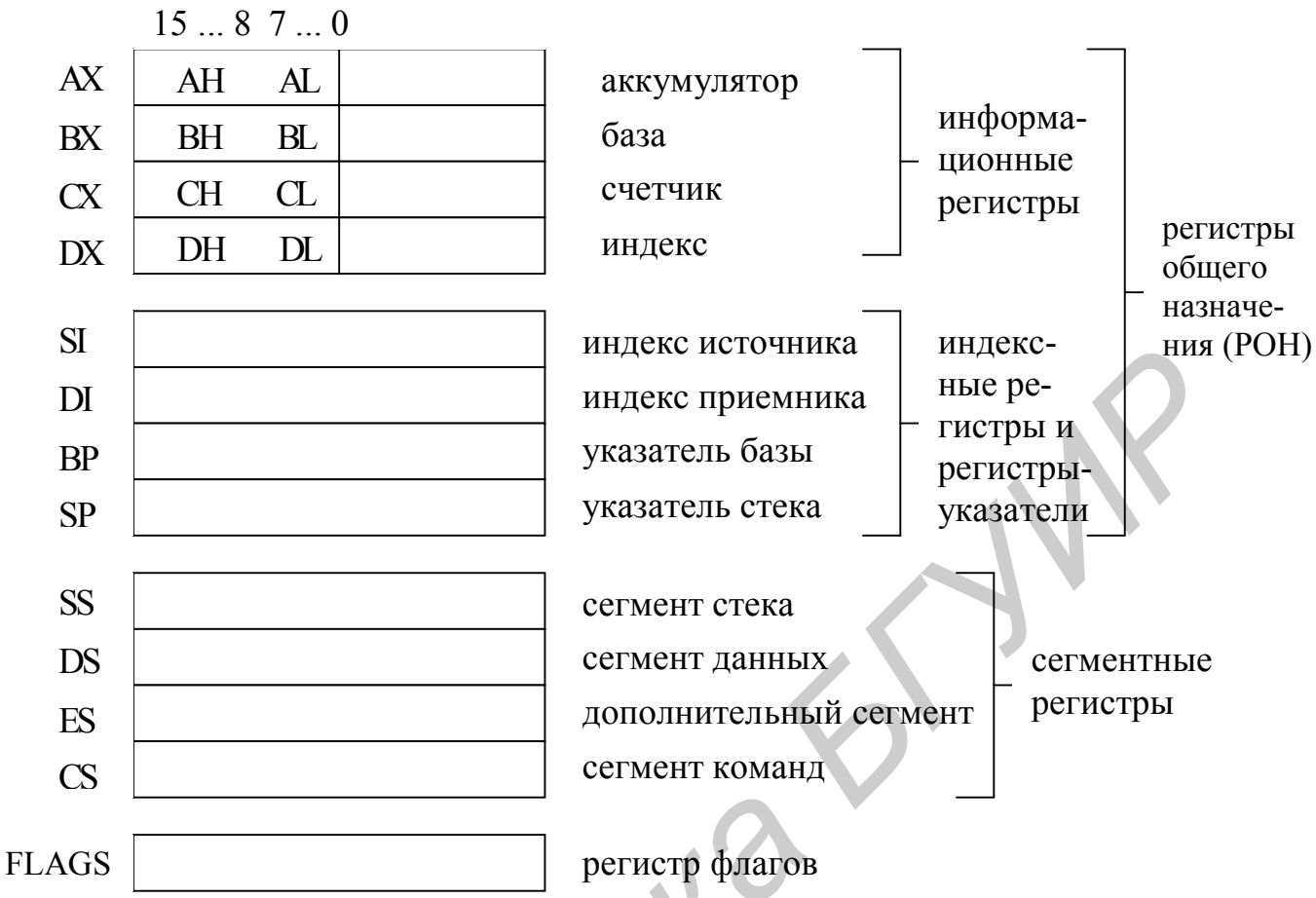



Рисунок 2.2 – Программно–доступные регистры МП 8086/88

– *регистр-счетчик* CX предназначен для организации циклов (хранит число повторений тела цикла);

– *регистр данных* DX служит в качестве вторичного аккумулятора для временного хранения данных и промежуточных результатов.

Четыре 16–разрядных *индексных* и *указательных регистра* (SI, DI, SP, BP) предназначены для хранения внутрисегментных смещений (они же могут участвовать и в выполнении арифметических и логических операций над двухбайтными словами). Содержимое *индексных регистров* SI и DI при определенных режимах адресации (возможно, совместно с содержимым регистра BX) используется при формировании относительного адреса для обращения к памяти. Код в регистрах SI и DI при выполнении команды может автоматически возрастать или уменьшаться в режиме автоиндексации, причем регистр SI применяется для индексации операнда, а регистр DI – результата. *Регистры–указатели* SP и BP используются для работы со стеком, который служит, в частности, для передачи аргументов подпрограмме, а также возврата значений и управления в основную программу (стек не организован аппаратно, а реализуется в указанном сегменте стека). Содержимое регистра SP (Stack Pointer – указатель стека) указывает на адрес элемента в вершине стека; регистр же BP используется аналогично регистру BX (в

том числе совместно с регистрами SI и DI), только указывает на адрес в стеке, а не в области данных (содержимым регистра BP можно отмечать дно стека, если его область пересекается с областью данных). Заметим, что индексные регистры и регистры–указатели могут применяться и по другому назначению, если в данный момент они не используются стандартным образом.

**2.2.2.5** *Сегментные регистры* играют важную роль в формировании физических (исполнительных) адресов ОП. В связи с тем, что внутренние регистры МП 8086/88 16–разрядные, напрямую можно адресовать только $2^{16}$=64 Кбайт памяти (минимальная адресуемая единица – байт); этого для реальных приложений явно недостаточно. Для расширения адресного пространства до 1 Мбайт (поддерживается 20–разрядным адресом) в МП 8086/88 используется *сегментная организация памяти*. В соответствии с этим 1 байт–адресное пространство логически делится на 64 К *параграфов* по 16 байт. Поэтому для хранения номера параграфа оказывается достаточным 16 разрядов, причем этот номер содержится в сегментном регистре. *Сегмент* представляет собой логическую область памяти, которая состоит из целого числа параграфов, начинается с какого–либо параграфа и содержимое которой можно адресовать 16–разрядным кодом. Следовательно, размер сегмента не может превышать 64 Кбайт (4 К параграфов). Исполнительный адрес формируется путем сложения двух составляющих:

а) 20–разрядного *адреса сегмента*, представляющего собой номер первого параграфа в сегменте, умноженный на 16 (умножение осуществляется путем сдвига содержимого сегментного регистра на четыре разряда влево; очевидно, что адрес сегмента всегда кратен 16);

б) 16–разрядного *относительного адреса* в рамках сегмента (относительный адрес внутри сегмента формируется в соответствии с заданным в команде режимом адресации из указателей адресов, хранящихся в РОНах, и содержимого адресного поля команды).

Таким образом, адрес сегмента имеет вид $XXXX0_{16}$, а относительный адрес – $XXXX_{16}$, где вместо X может быть записан любой 16–теричный символ от 0 до F (0,...,9, A, B, C, D, E, F).

МП имеет доступ одновременно к четырем сегментам:

– *сегменту команд (программному сегменту)* CS, содержащему последовательность команд программы;

– *сегменту данных* DS, хранящему данные для выполняемой программы;

– *дополнительному (экстракодовому) сегменту* ES, обычно используемому для запоминания промежуточных результатов, т.е. в качестве дополнительной рабочей памяти;

– *сегменту стека* SS, содержащему стек, в котором размещаются значения локальных переменных и чере который осуществляется связь по информации между основной программой и подпрограммой.

С точки зрения аппаратуры размер любого сегмента составляет 64 Кбайт. Меньший же размер может фиксироваться только в голове программиста путем ограничения области относительных адресов, за счет чего неиспользуемая часть сегмента может быть отведена под другой сегмент. Иногда удобно считать, что

память имеет *страничную организацию* с размером страницы, равным 64 Кбайт; тогда старшие четыре бита 20–разрядного адреса указывают страницу, а остальные – относительный адрес в рамках страницы.

**2.2.2.6** *Регистр флагов (признаков)* FLAGS предназначен для фиксации различных сигналов, вырабатываемых в МП в ходе интерпретации команд (к примеру, знака результата выполненной в АЛУ операции). Эта информация может использоваться для организации разветвлений в программе.

**2.2.2.7** *Регистр–указатель команд* IP (традиционное название – *счетчик команд* или *регистр адреса команд*) служит для хранения адреса очередной и формирования адреса следующей инструкции (в форме относительного адреса в пределах программного сегмента).

**2.2.2.8** *Блок формирования адресов* обеспечивает аппаратное формирование исполнительного адреса по описанным выше принципам (см. подпункт 2.2.2.5).

**2.2.2.9** Для указания местоположения команды в ОП используется пара регистров CS и IP – это записывается как CS:IP. Для адресации следующей команды в программе применяются три способа:

а) по мере загрузки и выполнения команды значение регистра IP увеличивается на ее длину (естественная последовательность выполнения команд друг за другом);

б) вводится команда перехода внутри текущего сегмента CS (NEAR–переход), которая изменяет значение только регистра IP;

в) вводится команда перехода между сегментами (FAR–переход), которая изменяет значение как регистра IP, так и регистра CS, т.е. появляется возможность реализовать переходы на расстояние более 64 Кбайт и выполнять программу, находящуюся в любом месте адресного пространства памяти).

**2.2.2.10** В процессоре Pentium многие регистры, аналогичные рассмотренным выше, имеют в своем обозначении приставку E, которая образована от слова extended (расширенный) и указывает на то, что в прежних моделях процессоров Intel эти регистры были 16–разрядными, а затем их разрядность была увеличена до 32 бит. РОНы совместно с регистрами сегментов используются в системе адресации процессора Pentium для задания виртуального адреса, который затем с помощью таблиц страниц отображается на физический адрес.

**2.2.3** Общие сведения о системе прерываний

**2.2.3.1** МП должен оперативно реагировать на различные события, происходящие в ПЭВМ в результате действий пользователя или без его ведома. В качестве примеров таких событий можно привести нажатие клавиши на клавиатуре (и другие события в ВУ), попытка деления на нуль, переполнение разрядной сетки, сбой питания и другие нарушения в работе оборудования, запланированные в программе обращения к ядру ОС и т.п. Необходимую реакцию на события обеспечивает система прерываний. Под *прерыванием* (interrupt) понимают сигнал, по которому процессор “узнает” о свершении в общем случае асинхронного (по сравнению с тактовой частотой процессора) события. Под *системой прерываний* понимают комплекс аппаратно–программных средств, обеспечивающий выявление и обработку прерываний.

Обработка прерываний сводится к приостановке исполнения текущей последовательности команд (программы), вместо которой начинает интерпретироваться другая последовательность инструкций, соответствующая данному типу прерывания и называемая *обработчиком прерывания* (IH – Interrupt Handler) или *процедурой обслуживания прерывания* (ISR – Interrupt Service Routine). После ее реализации исполнение прерванной программы может быть продолжено, если это возможно и/или целесообразно, что зависит от типа прерывания.

**2.2.3.2** В зависимости от источника возникновения прерывания делятся на следующие классы:

– внешние (аппаратные) прерывания;
– внутренние прерывания – захваты или исключения (exeption);
– программные прерывания.

**2.2.3.3** Сигналы *внешних прерываний* вырабатываются внешними устройствами, выставляются на системной шине и перед передачей в МП (на специальный вход прерывания) предварительно обрабатываются специальным устройством – *контроллером прерываний*. Так как эти прерывания возникают асинхронно по отношению к потоку инструкций прерываемой программы, то они воспринимаются процессором в ближайший следующий момент времени между выполнением двух соседних инструкций, т.е. в так называемой *точке наблюдения*. При этом система после обработки прерывания продолжает выполнение программы, уже начиная со следующей инструкции. Внешние прерывания делятся на *маскируемые* и *немаскируемые* (первые из них могут быть заблокированы или замаскированы, а вторые – нет). Немаскируемые прерывания предназначаются для реакции на “сверхважные” для системы события, например сбой по питанию.

**2.2.3.4** *Внутренние прерывания* происходят синхронно выполнению программы при появлении аварийной ситуации в ходе исполнения некоторой инструкции программы. Примерами *исключений* (непредопределенных событий в процессе) являются такие ошибки, как недопустимый код операции, обращения по несуществующему адресу, нарушение защиты памяти, попытка выполнить привилегированную инструкцию в пользовательском (исполнительском) режиме, обращение к устройству, отсутствующему в используемой конфигурации, а также ошибки, возникающие из–за необычных ситуаций с данными (попытка деления на нуль, переполнение при арифметических операциях, исчезновение порядка, потеря значимости).

**2.2.3.5** *Программные прерывания* отличаются от предыдущих двух классов тем, что они по своей сути не являются “истинными” прерываниями. Программное прерывание возникает при выполнении особой команды процессора (*системного вызова*), выполнение которой имитирует прерывание, т.е. переход на новую последовательность инструкций. Смысл программного прерывания заключается в том, чтобы вызвать из программы пользователя процедуру ОС с расширенными правами для получения сервисных услуг операционной системы.

**2.2.3.6** IBM–совместимые компьютеры поддерживают 256 типов прерываний, определяемых источником и причиной прерывания. Для их распознавания и разделения во времени каждому типу прерывания присваивается

определенный *код – вектор прерываний* (от 0 до 255) и *приоритет (уровень).* Число уровней может варьироваться от 4 до 15. Одновременное поступление в МП сигналов прерываний одного уровня приоритета обычно не допускается; поэтому прерывания от ВУ, которые могут функционировать одновременно, должны иметь различный уровень приоритета.

МП 8086 и процессор Pentium поддерживают векторную схему прерываний , с помощью которой может быть вызвано 256 процедур обработки прерываний (вектор имеет длину 1 байт, т.е. всего $2^8$=256 векторов). Соответственно *таблица прерываний* имеет 256 элементов, которые в реальном режиме работы процессора состоят из дальних адресов (CS:IP) этих процедур, а в защищенном режиме – из дескрипторов. Таблица прерываний реального режима, каждый элемент которой имеет длину 4 байта, всегда находится в фиксированном месте физической памяти – с начального адреса $00000_{16}$ по адрес $003FF_{16}$ (т.е. с нулевого по 1023–й байт). Старшие два байта элемента таблицы содержат адрес обработчика прерывания в виде номера первого параграфа кодового сегмента CS, а младшие два байта – относительный адрес IP (внутрисегментное смещение). Каждый элемент таблицы прерываний однозначно адресуется путем сдвига вектора прерывания на два разряда влево, что эквивалентно умножению на 4. В защищенном режиме таблица прерываний носит название IDT (Interrupt Descriptor Table) и может располагаться в любом месте физической памяти. Ее начало (32–разрядный физический адрес) и размер (16 бит) можно найти в регистре системных адресов IDTR; каждый из 256 элементов таблицы прерываний представляет собой 8–байтный дескриптор (напомним, что здесь речь идет о процессоре Pentium).

**2.2.3.7** Обработка прерывания состоит в следующем:

а) формируется его код (вектор), который поступает в МП;

б) в стеке сохраняется текущее состояние МП (содержимое регистров CS и IP) с целью возможного последующего восстановления;

в) по соответствующему вектору прерывания осуществляется переход на программу обработки прерывания данного типа;

г) выполняется собственно обработка прерывания;

д) восстанавливается исходное состояние МП, вследствие чего либо продолжается выполнение прерванной программы, либо завершается ее выполнение, либо происходит останов МП (в зависимости от типа прерывания).

**2.2.4** Интерфейс прикладного программирования

**2.2.4.1** Операционная система, кроме управления ресурсами системы, предоставляет пользователям удобный интерфейс к аппаратным средствам компьютера. С помощью ОС реальная машина, способная выполнять только небольшой набор элементарных действий, определяемых ее системой команд, превращается в *виртуальную машину* (ВМ), выполняющую широкий набор гораздо более мощных функций. Виртуальная машина тоже управляется командами, но это уже команды другого, более высокого уровня: удалить файл с определенным именем, запустить на выполнение некоторую прикладную программу, вывести текст из файла на печать и т.п..

**2.2.4.2** Прикладному программисту возможности ОС доступны в виде набора функций, составляющих *интерфейс прикладного программирования* (API–Application Programming Interface). От конечного пользователя эти функции скрыты за оболочкой алфавитно–цифрового (ОС MS–DOS) или графического (ОС типа Windows) интерфейса. Прикладные программисты используют в своих приложениях обращения к ОС, когда для выполнения тех или иных действий им требуется особый статус, которым обладает только операционная система. Например, в большинстве современных ОС все действия, связанные с управлением аппаратными средствами компьютера, может выполнять только ОС. Помимо этих функций прикладной программист может воспользоваться набором сервисных функций ОС, которые упрощают написание приложений.

**2.2.4.3** Приложения выполняют обращения к функциям API с помощью *системных вызовов*. Способ, которым приложение получает услуги ОС, очень похож на вызов подпрограмм. Информация, нужная ОС и состоящая обычно из идентификатора команды и данных, помещается в определенное место в памяти, в регистры и/или стек. Затем управление передается операционной системе (МП переходит в привилегированный режим), которая выполняет требуемую функцию и возвращает результаты через память, регистры или стеки. Если операция проведена неуспешно, то результат включает индикацию ошибки. На рисунке 2.3 представлен прикладной программный интерфейс ОС MS–DOS, включающий в себя функции DOS и функции BIOS (базовой системы ввода/вывода). Заметим, что прикладные программы всегда имеют возможность обращаться к устройствам компьютера напрямую, однако всегда, когда это возможно, следует использовать для этих целей функции API. Доступ к функциям API осуществляется через систему прерываний (см. пункт 2.2.3).

**2.2.4.4** DOS (сокращение для дисковой операционной системы, в данном случае – MS–DOS) представляет собой программу, управляющую работой компьютера и занимающая часть доступной ОП. Кроме пользовательского интерфейса, DOS предоставляют широкий спектр услуг, которые интенсивно используются практически всеми прикладными программами. Именно обращаясь к функциям DOS, прикладные программы выполняют чтение и запись в файлы, принимают нажатия клавиш, распределяют память, запускают другие программы и даже позволяют устанавливать и принимать время дня. Доступ к функциям DOS осуществляется через систему прерываний посредством системного вызова intdos; каждая функция имеет свой 16-теричный номер, например, функция DOS получения очередной нажатой на клавиатуре клавиши имеет номер 01H. Этот номер заносится в регистр AH, после чего выполняется прерывание BIOS 21H, которое произведет чтение символа с клавиатуры, возвращая в регистре AL код этого символа.

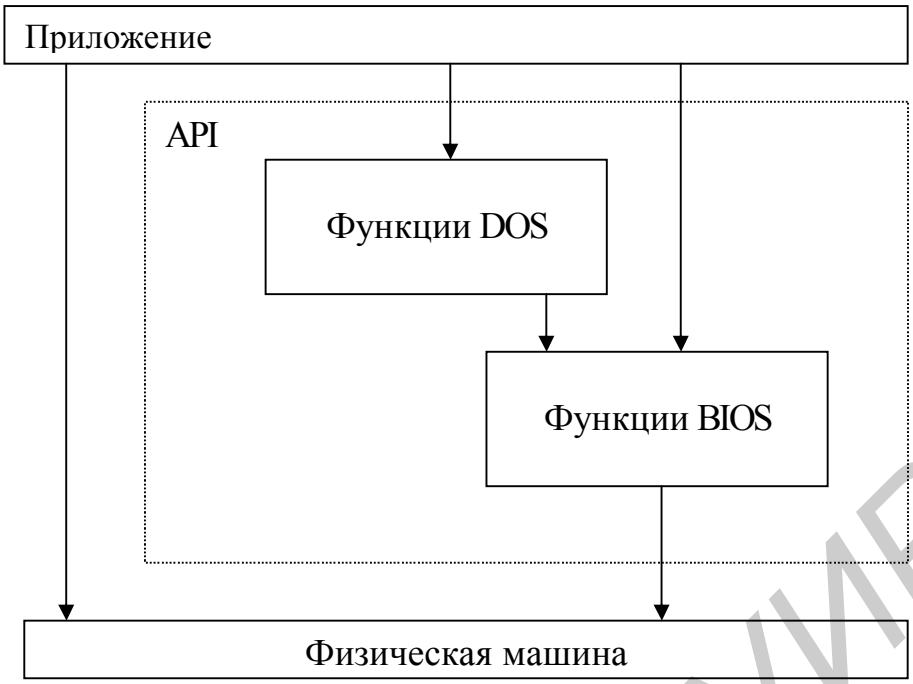



Рисунок 2.3 – Интерфейс прикладного программирования ОС MS–DOS

Иногда функции DOS не соответствуют вашим требованиям; в таких случаях необходимо обратиться к базовой системе ввода/вывода, или BIOS. В отличие от DOS и прикладного программного обеспечения, BIOS не загружается с диска и не занимает никакого участка в доступной оперативной памяти; BIOS хранится в ПЗУ (ROM) в той части адресного пространства прибора 8086, которая резервируется для системных функций. BIOS является программным обеспечением компьютера самого нижнего уровня; даже DOS использует функции BIOS для управления аппаратным обеспечением (см. рисунок 2.3). Поскольку BIOS, как и DOS, может маскировать различия между разными устройствами компьютера, лучше использовать функции BIOS, чем напрямую обращаться к аппаратному обеспечению ЭВМ. С другой стороны, если есть такая возможность, то предпочтительнее использовать функции DOS, а не BIOS, поскольку программы, использующие BIOS, могут вступать в конфликт с другими программами и теряют мобильность с точки зрения их переноса на другие модели компьютеров. Например, только запуская функции BIOS, вы можете устанавливать режим экрана, управлять цветами и т.д.

Прерывание BIOS 10H (это число 16 в 16–теричной системе) вызывается командой int86 и предоставляет прикладной программе большой видеосервис. Некоторые полезные функции этого прерывания приведены в таблице 2.1 [7]. Перед использованием в регистр AH заносится номер функции (первое шестнадцатеричное число в строке), остальные регистры используются каждой функцией отдельно.

Таблица 2.1 – Некоторые функции прерывания BIOS 10H

| Номер функции (в регистр AH) | Регистры | | Описание |
|---|---|---|---|
| | Входные данные | Выходные данные | |
| 0x00 (00H) | AL – режим (таблица 2.2) | – | Устанавливает режим монитора CGA |
| 0x01 (01H) | CH–начальная, CL – конечная строки (от 0 до 1fH); BH – номер видеостраницы (0 – текущая) | – | Устанавливает размер/форму курсора. Для подавления курсора в CH занести 0x20 (20H) |
| 0x02 (02H) | BH – номер видеостраницы; DH – строка, DL – колонка | – | Устанавливает курсор в новых координатах (считая от 0). Установка в строку 25 делает курсор невидимым |
| 0x03 (03H) | BH – номер видеостраницы | DH, DL – строка, колонка (координаты); CH, CL – начальная и конечная строки курсора (см.функцию 0x01) | Определяет положение и форму курсора |
| 0x06 (06H) | CH, CL (DH, DL) – строка, колонка верхнего левого (нижнего правого) угла; AL – число выдвигаемых строк; BH–атрибут пустых строк | – | Листает (очищает) окно вверх. Координаты углов считаются от нуля. Для очистки окна в AL заносится 0 |
| 0x07 (07H) | Аналогично функции 0x06 | – | Листает окно вниз, вдвигая пустые строки в верхнюю часть окна |
| 0x08 (08H) | BH – номер видеостраницы | AL – символ; AH – атрибут | Читает символ и атрибут в текущей позиции курсора |
| 0x09 (09H) | BH – номер видеостраницы;AL–символ; CX – счетчик; BL – атрибут | – | Пишет символ(ы) с заданным атрибутом в текущей позиции курсора |

Продолжение таблицы 2.1

| Номер функции (в регистр AH) | Регистры | | Описание |
|---|---|---|---|
| | Входные данные | Выходные данные | |
| 0x0a (0aH) | BH – номер видео-страницы; AL – символ; CX – счет-чик (сколько эк-земпляров символа записать) | – | Писать символ в текущей позиции курсора |
| 0x0b (0bH) | BH – 0; BL – цвет | – | Показывает бор-дюр экрана в уста-новленном цвете |
| 0x0c (0cH) | BH – номер видео-страницы; DX, CX – строка, ко-лонка; AL – значе-ние цвета | – | Писать графиче-скую точку |
| 0x0f (0fH) | – | AL – текущий ре-жим; AH – число текстовых колонок на экране; BH – номер активной страницы дисплея | Читать текущий видеорежим (см. функцию 0x00) |

Таблица 2.2 – Видеорежимы монитора CGA (EGA)

| Номер режима | Значение (тип и формат) |
|---|---|
| 0 | Текст 40x25 |
| 1 | Текст 40x25 |
| 2 | Текст 80x25 |
| 3 | Текст 80x25 |
| 4 | Графика 320x200 |
| 5 | Графика 320x200 |
| 6 | Графика 640x200 |

## 2.3 Порядок выполнения работы

**2.3.1** Загрузить Turbo C. В первой строке экрана можно увидеть главное меню TC [16].

**2.3.2** Для рассмотрения использования на практике функций DOS и BIOS создадим и выполним на языке Си небольшую программу. Неважно, что многие из вас незнакомы с языком программирования Си – в данной работе внимание будет

сосредоточено на другом – на получении навыков работы с функциями DOS и BIOS через программные прерывания. Язык Си здесь не более чем инструмент.

Если в окне редактора программ TC загружена какая-либо программа, то необходимо выбрать в главном меню (вход в главное меню через клавишу **F10**, выход – **Esc**) подрежим **New** режима **File** и нажать клавишу **Enter**. Окно редактора при этом очистится, и можно будет приступить к набору текста программы. В процессе ввода текста программы можно не набирать сообщения, заключенные в символы /* и */, – это комментарии.

Текст набираемой программы следующий:

```
        /*
        * Программа установки различных режимов монитора.
        * Режим монитора задается параметром командной строки.
        */
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
void main(int ac,char *av[]) /* int ac - число аргументов (включая имя
                                                        программы)        */
{                                   /*  char *av[] – указатель на
                                            массив с аргументами       */
union REGS r;
 int modes;
if(ac<2) /* Если параметров меньше двух, то сообщить */
{          /* об этом и завершить работу                          */
printf ( "\nMODESET режим\n\n") ;
printf ("Устанавливает монитор в заданный режим\n");
printf ("0 - текст 40x25\n");
printf ("1 - текст 40x25\n");
printf ("2 - текст 80x25\n");
printf ("3 - текст 80x25\n");
printf ("4 - графика 320x200\n");
printf ("5 - графика 320x200\n");
printf ("6 - графика 640x200\n");
exit(0);
}
/* Переменной modes присваивается значение первого параметра */
modes=atoi (av[1])%7;
/* Установка режима монитора */
r.h.ah=0;         /* В регистр AH заносится 0 – номер функции DOS */
r.hal=modes;   /*      В регистр AL заносится номер режима            */
 int86(0x10,&r,&r); /*      Вызывается прерывание 10H                 */
/*      Выдается сообщение на экран об установленном режиме       */
```

```
printf ("Режим монитора – %d,modes);
/* Ожидание нажатия клавиши */
r.h.ah=l;          /* В регистр АН заносится 1 – номер функции DOS */
intdos(&r,&r);  /*       Вызов функции DOS (прерывание 21H)        */
}
```

Примечания

1 Функция atoi(av[i]) преобразует строку символов в целый тип, где av[i] – i-й аргумент в обращении к программе.

2 При написании программ, обращаясь к байтам H или L, например регистра AX, следует писать соответственно r.h.ah или r.h.al, а при обращении ко всему регистру – r.x.ax.

**2.3.3** После ввода текста программы необходимо нажать клавишу **F2** (для сохранения текста) и ввести по приглашению имя файла (вместо NONAME.C, установленного по умолчанию). Расширение файла должно быть .C.

**2.3.4** Выберите подрежим **Arguments** режима **Options** главного меню, наберите в открывшейся строке любую цифру от 0 до 6 включительно (этим вы зададите программе аргумент, означающий номер режима) и нажмите клавишу **Enter**.

**2.3.5** Выберите подрежим **Compile_to_OBJ** режима **Compile** и нажмите клавишу **Enter**. По центру экрана откроется окно, где будет отображена информация по компиляции вашей программы. По ее завершении, если нет предупреждений и ошибок (сообщения на экране Warning: 0, Error: 0), выберите подрежим **Run** режима **Run** главного меню и нажмите клавишу **Enter**. После связывания объектного модуля вашей программы с библиотеками TC получится загрузочный модуль вашей программы, который будет сразу же запущен на выполнение – на темном экране вы увидите строку:

Режим монитора – 5 ,

если аргумент был установлен равным 5. Вернитесь в TC, нажав на любую клавишу.

**2.3.6** Включите строки программы (см. пункт 2.3.2)

```
r.h.ah=l;
intdos(&r,&r);
```

в комментарий и запустите ее заново (см. пункт 2.3.5). Отработав, программа сразу же вернется в TC. Вы убрали функцию ожидания ввода кода нажатой клавиши, и программа сразу возвращает управление той среде, откуда она была запущена. Для возврата к рабочей странице следует нажать комбинацию клавиш **Alt+F5**.

**2.3.7** Выберите подрежим **OS_Shell** режима **File** и нажмите на клавишу **Enter** – вы временно выйдите в систему MS-DOS. Наберите в приглашении системы строку, состоящую из имени вашей программы и параметра-режима через пробел. Например, если программа названа MODESET, то после набора строки

MODESET 0

и нажатия на клавишу **Enter** монитор перейдет в режим отображения текста по 40 символов в строке и 25 строк на экране. Для возврата в обычный текстовый режим следует набрать строку

MODESET 3 .

Для возврата в Turbo C следует ввести команду **EXIT**.

**2.3.8** Модифицировать программу так, чтобы она устанавливала курсор в заданные параметрами координаты экрана.

**2.3.9** Модифицировать программу так, чтобы она выводила на экран ваше имя, используя при этом функции BIOS.

**2.3.10** Модифицировать программу так, чтобы она рисовала на экране прямую линию (или фигуру) в режиме монитора 4 (см. таблицу 2.2).

**2.3.11** Модифицировать программу так, чтобы она убирала курсор с экрана, не меняя при этом режим монитора.

**2.3.12** Модифицировать программу так, чтобы она выводила на экран сообщение о текущем видеорежиме.

**2.3.13** Модифицировать программу так, чтобы она устанавливала вид курсора (параметрами командной строки задается начальная и конечная строки курсора).

**2.3.14** Продемонстрировать преподавателю правильность функционирования модифицированной программы.

## 2.4 Контрольные вопросы

**2.4.1** Что входит в типовую структуру компьютера фон-неймановской архитектуры?

**2.4.2** Какие ресурсы процессора доступны программисту? Перечислите состав программно–доступных регистров МП 8086/8088.

**2.4.3** Почему 16–разрядный процессор может адресовать только 1 Мбайт памяти и почему доступны только 640 Кбайт? Поясните суть сегментной адресации.

**2.4.4** Какие классы прерываний вы знаете?

**2.4.5** Поясните суть механизма прерываний.

**2.4.6** В чем заключается обработка прерывания?

**2.4.7** Как организовано использование прерываний в языках высокого уровня?

**2.4.8** В чем разница между функциями DOS и BIOS?

# 3 ОБСЛУЖИВАНИЕ ДИСКОВ (ЧАСТЬ I)

## 3.1 Цель работы

**3.1.1** Изучить системные утилиты обслуживания дисков.

**3.1.2** Освоить практическое применение системных утилит обслуживания дисков.

## 3.2 Справочные сведения

**3.2.1** Программы обслуживания дисков

**3.2.1.1** К системным программным средствам для обслуживания дисков обычно относят [2–4,12,14,15]: **Fdisk, Format, Disk Defragmenter, DriveSpace, ScanDisk, Recycle Bin,** и **Microsoft Backup.** Первые два подготавливают диск к работе, остальные обслуживают диски в процессе эксплуатации. Чтобы компьютер всегда был в “форме”, рекомендуется периодически проводить обслуживающие операции.

**3.2.1.2** Если программы **Disk Defragmenter, DriveSpace** и **Microsoft Backup** не установлены, то необходимо выполнить следующие действия: **Пуск – Настройка – Панель управления – Установка и удаление программ – Установка Windows.** В списке **Компоненты** установить флажок **Служебные**, далее кнопки **Состав** (установить требуемые флажки) – **ОК - Применить.**

**3.2.1.3** Чтобы посмотреть сведения по обслуживанию диска, следует вызвать контекст диска (**Правый щелчок по значку диска – Свойства**) и выбрать **Сервис**. В результате можно видеть, когда последний раз диск обслуживался, а также рекомендации по обслуживанию.

**3.2.2** Программа **ScanDisk**

**3.2.2.1** Программа проверки диска **ScanDisk** — это диагностическая утилита, которая следит за состоянием жестких дисков и либо сама устраняет появляющиеся сбои, либо сообщает о них Вам. После установки Windows ее можно использовать как для обычных, так и для сжатых дисков. В Windows 9x существуют две версии **ScanDisk**: **SCANDSKW.EXE —** новое графическое Windows-приложение, которое запускается из меню **Пуск**, и **SCANDISK.EXE** — вариант для MS DOS, содержащийся на системной дискете и в папке Windows Command.

**3.2.2.2** Программа проверки диска **ScanDisk** может работать с любым жестким диском (НЖМД) или дискетой (НГМД), включая сжатые утилитой уплотнения диска **DriveSpace (**или **DoubleSpace**). Эта программа способна обнаружить и исправить следующие логические ошибки:

– ошибки в таблице размещения файлов (FAT);

– ошибки, связанные с длинными именами;

– в структуре файловой системы – потерянные кластеры, файлы с общими кластерами;

– в структуре дерева каталогов;

– на сжатых дисках – ошибки, связанные с заголовком тома, структурой файлов тома и структурой сжатия.

**3.2.2.3** *Таблица размещения файлов* (FAT – File Allocation Table) – это структура данных, в которой хранится информация о физическом расположении и принадлежности каждого кластера диска. *Кластер*, как единица дискового пространства, представляет собой наименьшую группу секторов, которую ОС может выделить файлу. *Сектор*, в свою очередь, состоит из поля данных (размером 512 байт) и поля служебной информации, ограничивающей и идентифицирующей его. *Потерянный кластер* — это кластер, который не занят ни одним файлом и который не помечен как свободный. *Файлы с общими кластерами* – это файлы, содержащие участки, которые по ошибке были выделены более чем одному файлу.

**3.2.2.4** Программу **ScanDisk** можно использовать для поиска физических ошибок диска (плохих секторов), при этом она не выполняет физическое восстановление поверхности диска, но может перенести данные из сбойных секторов в нормальные.

**3.2.2.5** Для запуска программы **ScanDisk** следует проделать следующие действия: **Пуск – Программы – Стандартные – Служебные программы – Проверка диска** или **Мой компьютер – Правый щелчок по значку проверяемого диска – Свойства – Сервис – Выполнить проверку…** В диалоговом окне программы **ScanDisk** выберите диск для проверки (если он не был выбран ранее), а нажав и удерживая нажатой клавишу **Ctrl**, можно выбрать для проверки несколько дисков.

**3.2.2.6** Если вы хотите выполнить только логическую проверку диска, то установите переключатель **Стандартная**, а если надо проверить еще и физическое состояние носителя, то включите опцию **Полная**. Кнопка **Дополнительно** (как при стандартной, так и при полной проверке) позволяет указать некоторые дополнительные параметры настройки, используемые для проверки логических ошибок. Поясним некоторые из них.

**3.2.2.7** Если программа **ScanDisk** обнаружит файлы с общими кластерами (параметр **Файлы с общими кластерами**), то она по умолчанию (установлен переключатель **Делать копии)** создаст отдельную копию каждого из вовлеченных в такое пересечение файлов и попутно удалит пересекающиеся ссылки в таблице размещения файлов. В большинстве случаев данные, располагающиеся в общем кластере, принадлежат какому-то одному файлу, и после того, как **ScanDisk** завершит диагностику, вы сможете удалить из файлов лишние данные при помощи обычного редактора. Впрочем, вы можете настроить **ScanDisk** и так, чтобы она просто удаляла (переключатель **Удалять**) или игнорировала (переключатель **Пропускать**) пересекающиеся файлы.

**3.2.2.8** По умолчанию (установлен переключатель **Преобразовывать в файлы** параметра **Потерянные цепочки кластеров**) программа **ScanDisk** собирает все потерянные кластеры в файлы, помещает их в папку верхнего уровня (корневой

каталог) того диска, на котором эти кластеры были обнаружены, и присваивает таким файлам имена типа File0000, File0001 и т.д. Эти файлы можно в дальнейшем просмотреть и решить, содержат они нужные данные или нет. Если вы хотите, чтобы программа **ScanDisk** лишь освободила все потерянные кластеры, то необходимо установить переключатель **Освобождать**.

**3.2.2.9** Если на компьютере имеется диск, уплотненный с помощью программы **DriveSpace** (или **DoubleSpace**), программа **ScanDisk** вначале проверяет неуплотненный диск, на котором этот сжатый диск расположен. Неуплотненный диск называется несущим и обычно невидим (его имя не присутствует в перечне дисков). Ошибки на уплотненном диске часто вызваны ошибками на несущем диске. Поэтому, если не установлен флажок **Проверить сперва несущий диск**, программа **ScanDisk** не будет проверять несущий диск.

**3.2.2.10** По умолчанию программа **ScanDisk** по завершении работы выводит на экран диалоговое окно с информацией об обнаруженных ошибках, а также записывает соответствующую информацию в файл протокола **Scandisk.log**. Этот файл помещается в корневой каталог проверявшегося диска и по умолчанию заменяет предыдущий файл протокола. Вы можете изменить такой режим вывода протокола при помощи групп переключателей **Выводить итоговые результаты** и **Файл протокола**.

**3.2.2.11** Переключатель **Полная** диалогового окна программы **ScanDisk** позволяет проверять и физические ошибки. При выборе этого режима становится доступной кнопка **Настройка…,** которая позволяет упростить тестирование в режиме полной проверки, например, вы можете ограничить проверку системной областью диска или областью данных, а также отказаться от проверки на запись.

**3.2.3** Дефрагментация дисков

**3.2.3.1** Когда вы записываете файлы на только что отформатированный диск, ОС помещает данные в расположенные один за другим кластеры диска. Но когда вы впоследствии начнете удалять ставшие ненужными файлы, то эта стройная структура будет нарушена и файлы начнут занимать несмежные кластера, т.е. будут фрагментированными. Таким образом, с течением времени, в процессе создания и удаления большого числа файлов, будет повышаться вероятность того, что все большее число файлов будет фрагментировано. Это не нарушает целостности данных в файле, однако производительность компьютера падает, так как чтение фрагментированных файлов выполняется дольше, чем последовательных.

**3.2.3.2** Для ускорения доступа к файлам используют программу дефрагментации **Disk Defragmenter**. Она упорядочит файлы на диске, располагая каждый из них в последовательной цепочке секторов. Этой программой можно обработать любой несжатый локальный диск или дискету, а также любой локальный диск, сжатый программой **DriveSpace** (или **DoubleSpace**). Дефрагментатор не работает с дисками “только для чтения”, заблокированными дисками, сетевыми дисками, а также дисками, созданными командами ASSIGN, SUBST или JOIN, и, как правило, не работает с дисками, сжатыми “неродными” утилитами.

**3.2.3.3** Для запуска дефрагментатора следует проделать следующие действия: **Пуск – Программы – Стандартные – Служебные программы – Дефрагментация**

**диска** или **Мой компьютер – Правый щелчок по значку проверяемого диска – Свойства – Сервис – Выполнить дефрагментацию…** Можно также в пункте главного меню **Выполнить** набрать команду **defrag** и в открывшемся окне выбрать требуемый диск.

**3.2.3.4** Перед запуском дефрагментатора (кнопка **ОК**) можно щелкнуть кнопку **Настройка…** с целью настройки следующих параметров дефрагментации:

– переместить файлы программ для ускорения их запуска;

– задать проверку диска на наличие ошибок.

Выбранные параметры можно сохранить и использовать в дальнейшем как параметры по умолчанию, установив переключатель **При каждой дефрагментации диска**.

**3.2.3.5** После запуска дефрагментатора, при выборе кнопки **Сведения,** открывается окно с полной текущей информацией о процессе дефрагментации. Если в этом окне выбрать кнопку **Легенда,** то будут выданы пояснения к информации в окне. Однако в таком режиме процесс дефрагментации замедляется. Чтобы его максимально ускорить, надо свернуть окно.

Примечание – Для ускорения операций по обслуживанию дисков не следует запускать другие программы при выполнении **ScanDisk** и **Disk Defragmenter**. Дело в том, что эти утилиты повторно инициализируют себя всякий раз, когда на диск что-то записывается, так как они должны иметь дело с текущим состоянием диска.

**3.2.4** Сжатие дисков программой **DriveSpace**

**3.2.4.1** Windows 9x использует жесткий диск компьютера в качестве расширения оперативной памяти – когда основной памяти не хватает, данные перекачиваются на диск и затем считываются с него по мере надобности. Подобное использование жесткого диска называется организацией виртуальной памяти. В Windows 9x реализован 32-разрядный доступ в защищенном режиме, а размер файла подкачки определяется потребностями Windows. Таким образом, вы получаете максимальную производительность без чрезмерного расхода дискового пространства. Способность Windows динамически изменять размер файла подкачки предполагает, что пользователь не должен решать, каким будет размер этого файла. Подразумевается, что наилучшая производительность будет достигнута, если вы позволите Windows самостоятельно управлять размером файла подкачки. Заметим, что по умолчанию Windows создает файл подкачки на том же диске, где находятся ее системные файлы.

**3.2.4.2** Для того чтобы виртуальная память использовалась максимально эффективно, на жестком диске должно быть достаточно свободного места (желательно не фрагментированного) для файла подкачки. Если объем свободного пространства не превышает 20 Мбайт, то производительность системы может заметно снизиться. Для того чтобы избежать этого, можно:

– удалить приложения и компоненты Windows, которыми вы не пользуетесь;

– сократить максимально возможный размер “корзины”;

– освободить место на диске при помощи программы уплотнения диска **DriveSpace**.

**3.2.4.3** Под сжатием дисков или их уплотнением подразумевается удаление избыточной информации. Например, в нашем текстовом файле между заглавием раздела 3 и подраздела 3.1 расположена пустая строка. При хранении файла такую строку можно представить не в “натуральном” виде, а как, например, запись текста: “70 пробелов”.

Программа сжатия дисков **DriveSpace** позволяет:

– сжимать и распаковывать гибкие диски, сменные носители, жесткие диски;

– работать с дисками, сжатыми **DriveSpace** или **DoubleSpace**;

– создавать новый сжатый диск из свободного пространства на несжатом диске.

Примечания

1 Если диск компрессировался (сжимался) программой **DoubleSpace** в реальном режиме, то при работе с ним в Windows 9x в некоторых случаях система будет предлагать сделать перезагрузку, что и надо делать.

2 Программа **DriveSpace** не сжимает диски с файловой системой FAT32!

**3.2.4.4** Программа уплотнения диска **DriveSpace** увеличивает емкость жестких дисков и дискет. Однако сжатый диск — не настоящее дисковое устройство, хотя и кажется таковым. На самом деле он располагается на физическом, жестком диске в виде *файла сжатого тома* (CVF – Compressed Volume File). CVF — это файл, создаваемый программой **DriveSpace** и имеющий атрибуты “только для чтения”, “скрытый” и “системный”. Каждый такой диск располагается на несжатом диске — хост-диске (или несущем диске). CVF-файл хранится в корневом каталоге с именем типа DRVSPACE.000 или DBLSPACE.000. Когда вы сохраняете файл на сжатом диске, то программа **DriveSpace** сжимает такой файл и записывает его в CVF-файл без каких-либо действий с вашей стороны. При считывании со сжатого диска файл автоматически распаковывается. В результате вы работаете с файлами как обычно, а объем свободного пространства на диске заметно увеличивается.

Примечание – Самым главным недостатком программы **DriveSpace** является не незначительное снижение производительности, а опасность нарушения целостности данных. Поэтому, чтобы не потерять файлы на сжатом диске, не экспериментируйте с CVF-файлом!

**3.2.4.5** Обычно CVF-файлы содержат больший объем информации, чем занимаемое им место на хост-диске. Программа **DriveSpace** назначает сжатому диску букву, так что к нему можно обращаться как к обычному дисковому устройству. Хост-диску присваивается своя буква (но это дисковое устройство может быть скрыто).

**3.2.4.6** Со сжатым диском работают так же, как и раньше. Кроме того, **DriveSpace** создает новый несжатый диск – хост-диск, на котором он и сохраняет CVF-файл. Если хост-диск содержит помимо CVF-файла дополнительное свободное пространство, его можно использовать для хранения файлов, которые нельзя сжимать.

**3.2.4.7** Новый сжатый диск можно создать и из свободного пространства на несжатом логическом диске, который является частью жесткого несменного диска.

**3.2.4.8** Реестр можно размещать на сжатом диске.

**3.2.4.9** Файл подкачки может находиться на сжатом диске, если тот управляется драйвером **drvspace.vxd**. Программа **DriveSpace** отмечает файл подкачки как не подлежащий сжатию и для уменьшения фрагментации помещает его в CVF-файл как последний файл. Такое размещение позволяет также расширять файл подкачки и на сжатом диске.

Вообще, *файл подкачки*, или *страничный файл* (win386.swp) используется при работе с виртуальной памятью для обмена страниц и располагается в корневом каталоге Windows.

Если необходимо изменить размер сжатого диска, а программа **DriveSpace** обнаружит в конце области размещения секторов файл подкачки, то будет предложено перезагрузить компьютер.

**3.2.4.10** Для запуска программы **DriveSpace** следует проделать следующие действия: **Пуск – Программы – Стандартные – Служебные программы – Сжатие данных** или **Правый щелчок по значку сжимаемого диска – Свойства – Сжатие**. Для той же цели можно выбрать команду **Выполнить** Главного меню и ввести **drvspace.** В результате откроется окно **DriveSpace,** где предлагается выбрать сжимаемый диск и три пункта меню: **Диск**, **Дополнительно** и **Справка** (рисунок 3.1).

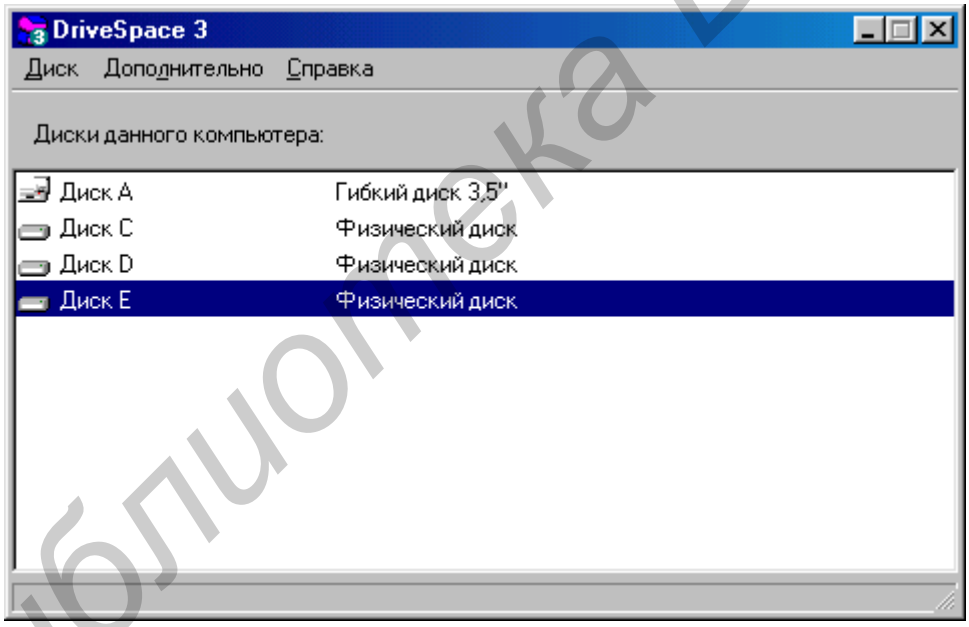



Рисунок 3.1 – Диалоговое окно программы

**3.2.4.11** Если выбрать команду **Сжать** меню **Диск**, то появится окно **Сжатие диска** с информацией о сжимаемом диске и о прогнозируемых результатах сжатия (прогноз, естественно, приближенный). Выбор кнопки **Настройка** позволяет задать имя (букву) несущего диска (хост-диска), оставить на хост-диске какой-то объем свободного пространства и указать, следует ли скрыть хост-диск.

После выбора кнопки **Запуск** будет предложено создать загрузочный диск, что желательно сделать при отсутствии такового.

На следующем шаге система предложит создать резервные копии файлов (кнопка **Архивировать файлы)**.

После выбора команды **Уплотнить сейчас** DriveSpace проверяет диск на наличие ошибок и начинает сжатие. Этот процесс занимает от нескольких минут до нескольких часов. Программа **DriveSpace** проверяет и перепроверяет целостность данных в процессе сжатия, поэтому эта процедура безопасна.

Если на диске есть открытые файлы, программа **DriveSpace** предложит закрыть их. В случае дисков, на которых всегда есть открытые файлы (например, где установлена Windows 9x или имеется файл подкачки), программа **DriveSpace** перезапустит компьютер и станет, пока идет сжатие, пользоваться компактной версией Windows. Для этого создается каталог **FILESAVE.DRV**, содержащий

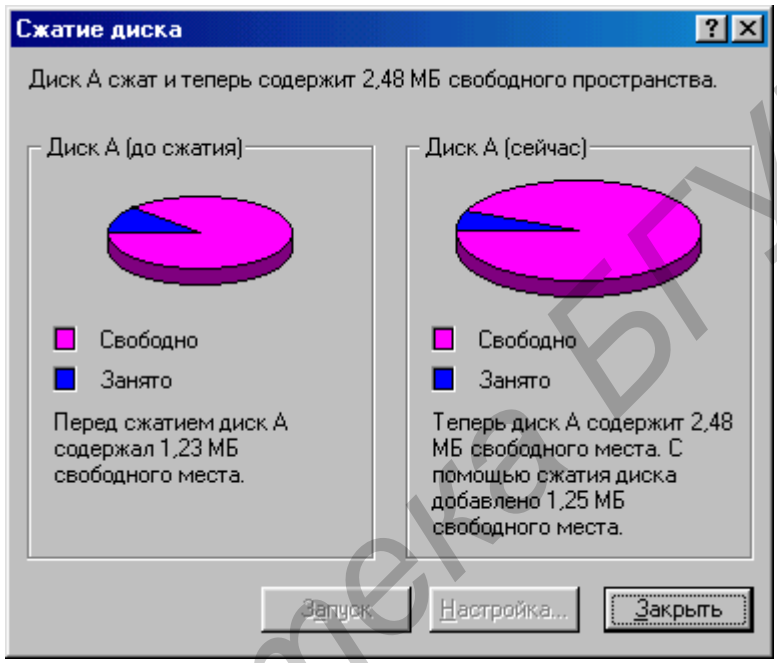



Рисунок 3.2 – Сообщение о результатах сжатия дискеты программой **DriveSpace**

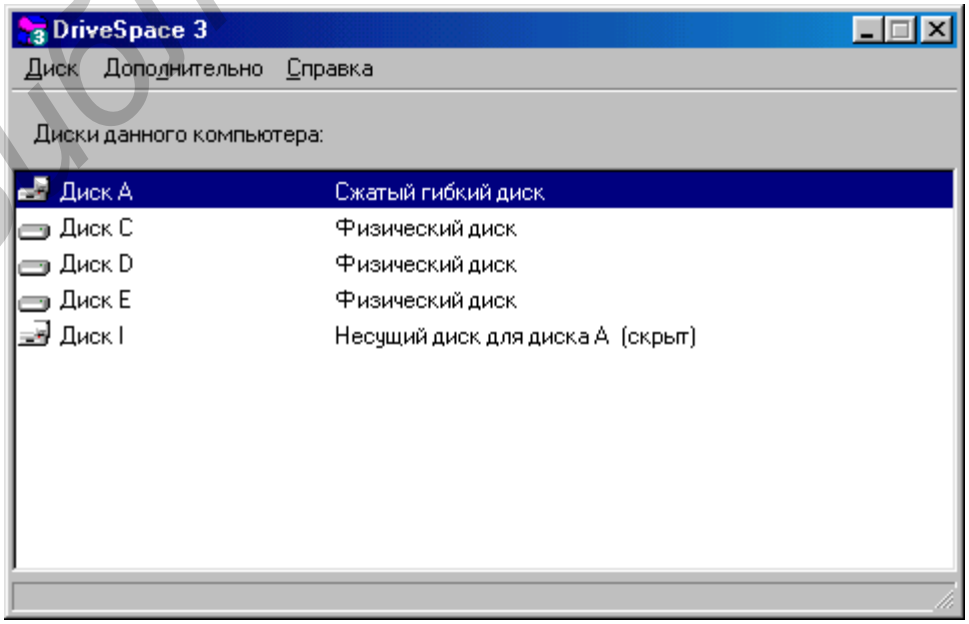



Рисунок 3.3 – Диалоговое окно программы **DriveSpace** после сжатия дискеты

необходимые системные файлы. После сжатия компьютер снова перезагружается.

**3.2.4.12** Вместо сжатия всего диска, программа **DriveSpace** может сжать только свободное пространство, при этом оно превратится в новый сжатый диск под неиспользованным до этого именем (буквой). Для создания нового (пустого) сжатого диска следует в окне **DriveSpace** выбрать диск (см. рисунок 3.1), а в меню **Дополнительно** выбрать команду **Создать пустой…**. В появившемся окне можно установить желаемые размеры формируемого диска, после чего нажать кнопку **Запуск.**

**3.2.4.13** Для распаковки сжатого диска в окне **DriveSpace** выбрать сжатый диск и выполнить команду **Диск – Распаковать…** После запуска кнопкой **Запуск** в случае, если на хост-диске недостаточно свободного пространства для размещения распакованных файлов, программа **DriveSpace** предупредит об этом. А если все в порядке, то программа **DriveSpace** предложит создать резервные копии файлов и приступит к распаковке.

**3.2.4.14** Командой **Диск – Сжать…** окна **DriveSpace** вы можете сжать целую дискету, но сжать свободное пространство дискеты нельзя. Сжатые дискеты становятся доступными системе сразу же по завершении процесса сжатия или всякий раз, когда вы вставляете их в дисковод. Разумеется, при этом на компьютере должна быть утилита **DriveSpace** или **DoubleSpace**. Сжатый гибкий диск можно применять для хранения данных или переноса их с одного компьютера на другой. На рисунке 3.2 изображено итоговое окно **Сжатие диска** после сжатия дискеты.

Кроме того, на компьютере появится диск с новой буквой (рисунок 3.3). Она назначается хост-диску, содержащему CVF-файл дискеты. Хотя и новая, и исходная буквы диска относятся к одному и тому же физическому дисководу, для доступа к нему используется только исходная буква.

Для отключения автоматического монтирования (присоединения) дисков в меню **Дополнительно** окна **DriveSpace** (см. рисунок 3.1) следует выбрать команду **Настройка…** и сбросить флажок **Автоматически присоединять новые сжатые диски**. Если автоматическое монтирование (присоединение) отключено, то для работы со сжатыми дискетами их придется монтировать (присоединять) явно. Для монтировки сжатого гибкого диска в **DriveSpace** необходимо вставить дискету в дисковод и в меню **Дополнительно** окна **DriveSpace** (см. рисунок 3.1) выбрать команду **Присоединить…**. Файлы на этом диске доступны до смены дискеты или перезагрузки компьютера. Признаком неприсоединенного сжатого диска является появление в его корневом каталоге файла **Readthis.txt**.

Для форматирования сжатой дискеты необходимо выбрать в главном диалоговом окне **DriveSpace** дисковод, в который она установлена, а затем выполнить команду **Диск – Форматировать…** Заметим, что эта команда доступна только для сжатых дискет. Кроме того, вы не сможете отформатировать присоединенную сжатую дискету при помощи стандартной команды **format** (предварительно ее надо распаковать или можно применить команду **format** для неприсоединенной сжатой дискеты).

**3.2.4.15** С помощью программы **DriveSpace** можно измененить размер свободного пространства на сжатом диске и коэффициент сжатия.

Оценка свободного пространства осуществляется с применением предполагаемого коэффициента сжатия, который можно установить в явном виде и таким образом задать степень сжимаемости файлов.

Обычно устанавливают предполагаемый коэффициент сжатия по фактическому коэффициенту сжатия файлов, уже имеющихся на сжатом диске, но его можно занижать или завышать в зависимости от того, какие файлы будут сжиматься: плохо сжимаемые файлы (например exe) или сильно сжимаемые (например текстовые). Изменение предполагаемого коэффициента сжатия не влияет на степень сжатия программой **DriveSpace**, а только на оценку свободного пространства.

Для изменения предполагаемого коэффициента сжатия необходимо в меню **Дополнительно** окна **DriveSpace** выбрать команду **Степень сжатия…** (см. рисунок 3.1).

Кроме того, на сжатом диске или его хост-диске можно скорректировать размер свободного пространства. Это эквивалентно изменению размера сжатого диска и его CVF-файла.

Для изменения размера свободного пространства на диске надо в меню **Диск** выбрать команду **Изменить размер…**.

Примечание – Если свободное пространство сильно фрагментировано, объем пригодного для использования свободного пространства может оказаться много меньше прогнозируемого и недостаточным для размещения файлов. Чтобы избежать этой проблемы, надо регулярно запускать **Disk Defragmenter**.

## 3.3 Порядок выполнения работы

**3.3.1** Работа с программой **ScanDisk**

**3.3.1.1** Выполнить стандартную проверку диска **D:**, установив следующие параметры:

– всегда выводить итоговые результаты;
– заменить файл протокола;
– делать копии файлов с общими кластерами;
– преобразовывать потерянные цепочки в файлы;
– проверить наличие ошибок в именах файлов;
– проверить наличие ошибок в дате и времени создания файлов;
– проверить вначале несущий диск.

**3.3.1.2** Выполнить полную проверку дискеты:

– с проверкой записи;
– без проверки записи.

**3.3.1.3** Результаты всех проверок (диалоговые окна с информацией об обнаруженных ошибках) поместить в текстовый файл отчета и прокомментировать. Сделать соответствующие выводы.

**3.3.2** Работа с программой **Disk Defragmenter**

**3.3.2.1** Дефрагментировать диск **D:**, установив следующие параметры дефрагментации:

– переместить файлы программ для ускорения их запуска;

– проверить диск на наличие ошибок;

– использовать установленные параметры при каждой дефрагментации диска.

Окна (стартовое, промежуточные и итоговое) поместить в файл отчета.

**3.3.2.2** Дефрагментировать дискету с параметрами дефрагментации из предыдущего подпункта. Окна (стартовое, промежуточные и итоговое) поместить в файл отчета.

**3.3.2.3** После каждого запуска дефрагментатора войти в режим **Сведения** и наблюдать процесс.

**3.3.3** Работа с программой **DriveSpace**

**3.3.3.1** Образовать на свободном пространстве диска **D:** пустой сжатый диск и записать в текстовый файл стартовое и итоговое окна **Сжатие диска**. В случае, если это сделать будет невозможно, скопировать в текстовый файл соответствующее окно с известием об этом.

**3.3.3.2** Поместить какую-нибудь информацию на сжатый диск и продемонстрировать работу со сжатым диском преподавателю.

**3.3.3.3** Распаковать сжатый диск и записать в текстовый файл итоговое окно **Распаковка диска**.

**3.3.3.4** Сжать дискету и записать в текстовый файл отчета стартовое и итоговое окна **Сжатие диска**.

**3.3.3.5** Убедиться в работоспособности сжатой дискеты.

**3.3.3.6** Восстановить дискету в исходное состояние и после распаковки записать в текстовый файл итоговое окно **Распаковка диска**. Сделать выводы по итогам работы с программой **DriveSpace**.

## 3.4 Контрольные вопросы

**3.4.1** Что относят к средствам для обслуживания дисков?

**3.4.2** Какие ошибки устраняет **ScanDisk**, а какие только обнаруживает?

**3.4.3** Что такое FAT?

**3.4.4** Чем отличается полная проверка диска от стандартной?

**3.4.5** Что такое файлы с общими кластерами?

**3.4.6** Что такое “Потерянные цепочки”**?**

**3.4.7** Прокомментировать отчет о результатах работы **ScanDisk**.

**3.4.8** Прокомментировать окно **Дополнительные параметры** программы **ScanDisk**.

**3.4.9** Прокомментировать окно **Режим проверки поверхности** программы **ScanDisk**.

**3.4.10** Что такое фрагментация диска?

**3.4.11** Почему во время работы **ScanDisk** и **Disk Defragmenter** не рекомендуется запускать другие программы?

**3.4.12** Прокомментировать окно **Легенда** программы **Disk** Defragmenter.

**3.4.13** Что позволяет выполнить программа **DriveSpace**?

**3.4.14** Что такое монтировка диска?

**3.4.15** Что такое сжатый диск?

**3.4.16** Что такое хост-диск?

**3.4.17** Что такое файл подкачки?

**3.4.18** Как работает программа **DriveSpace**, если на диске есть открытые файлы?

**3.4.19** От чего зависит степень сжатия диска?

**3.4.20** Всегда ли сжатый диск вместит обещанный объем информации?

# 4 ОБСЛУЖИВАНИЕ ДИСКОВ (ЧАСТЬ II)

## 4.1 Цель работы

**4.1.1** Изучить системные утилиты обслуживания дисков.

**4.1.2** Освоить практическое применение системных утилит обслуживания дисков.

## 4.2 Справочные сведения

**4.2.1** Системные средства защиты информации

Рассматриваемую в данной лабораторной работе группу утилит и команд, наряду с другими средствами, иногда относят к системным средствам защиты информации [4,12,14,15]. Основными командами являются:

– **Diskcomp** (внешняя) – сравнение содержимого двух дискет;

– **Copy** (внутренняя) – копирование файлов;

– **Xcopy** (внешняя) – копирование файлов с сохранением структуры каталогов;

– **Sys** (внешняя) – копирование системных файлов на диск;

– **Diskcopy** (внешняя) – копирование дискеты (точная физическая копия).

Не будем подробно останавливаться на командах. Отметим лишь команду **Diskcopy**, которая для нормальной работы требует такой же дискеты, как и исходная. Поскольку эта команда создает точную физическую копию, то нередко позволяет обходить защиту расположенного на ней программного продукта от копирования.

Примечание – Для получения краткой справки по программе или команде DOS вы можете ввести эту команду (имя программы) с параметром /?.

**4.2.2 Recycle Bin (Корзина)**

**4.2.2.1** Каким бы способом вы ни уничтожили объект (папку, файл, ярлык) или группу объектов, Windows выведет на экран окно-приглашение и попросит подтвердить ваше намерение (это является одной из форм защиты информации). Если вы не хотите, чтобы Windows запрашивала подтверждение при удалении папок или файлов, сбросьте флажок **Запрашивать подтверждение на удаление** в нижней части списка свойств **Корзины**. Дополнительная защита вашей информации обеспечивается тем, что объекты, удаленные из папок жесткого диска или с поверхности рабочего стола, автоматически помещаются в **Корзину**, из которой впоследствии их можно восстановить.

**4.2.2.2** В течение некоторого времени после удаления объект хранится в **Корзине** (за исключением способа удаления **Выделить объект – Shift+Del**). Вы можете восстановить ранее уничтоженные объекты, и при этом они вернутся в те папки, из которых были удалены.

Для восстановления объекта, попавшего в корзину, отметьте его и выберите команду **Восстановить** в меню **Файл** или щелкните по объекту правой кнопкой мыши и выполните команду **Восстановить** из меню объекта. Эта команда перенесет выбранный объект в ту папку, из которой он был удален. Если такой папки уже нет, то Windows попросит разрешения создать ее заново. При восстановлении объекта вы можете поместить его и в другую папку. Для этого выделите нужный объект и выберите в меню объекта или в меню **Правка** команду **Вырезать**. После этого перейдите к той папке, в которую вы намерены поместить восстанавливаемый объект, и выполните команду **Вставить** из меню **Правка** окна папки-приемника.

**4.2.2.3** Следует также помнить о следующем:

– при восстановлении файла, находившегося в удаленной папке, вначале будет восстановлена сама эта папка;

– файлы, удаленные в окне сеанса MS-DOS, файлы, удаленные с сетевых дисков, а также файлы, удаленные со съемных носителей (например дискет), в папку **Корзина** не помещаются. Такие файлы удаляются сразу без возможности восстановления;

– некоторые прикладные программы снабжены своими собственными командами удаления файлов. Если вы воспользуетесь для уничтожения командой приложения, то файл может и не попасть в папку **Корзина**;

– чтобы восстановить только часть файлов из папки **Корзина**, выбирайте их, удерживая клавишу **Ctrl**, а затем выберите в меню **Файл** команду **Восстановить**.

Для восстановления всех всех файлов папки, которую вы случайно удалили, упорядочите содержимое **Корзины** по параметру **Вид – Упорядочить значки – по исходному размещению**. Это даст вам возможность увидеть все файлы, которые ранее находились в одной папке. Затем выделите эти файлы и выберите в меню **Файл** команду **Восстановить**.

**4.2.2.4** Несмотря на то что у вас только один значок папки **Корзина**, Windows поддерживает отдельные корзины для каждого жесткого диска на компьютере. По умолчанию размер каждой из них составляет 10% объема того жесткого диска, на котором она хранится. Чтобы она не занимала место, ее периодически надо «чистить». Но этот объем не пропадает зря. Просто суммарный размер всех файлов в корзине не может быть больше этих 10%. При превышении этого объема Windows начинает удалять файлы из корзины по-настоящему, при этом первыми удаляются наиболее старые файлы. При этом система не предупреждает о переполнении корзины!

Эту проблему можно решить увеличением пространства, занимаемого корзиной. Для изменения объема корзины необходимо сделать следующие действия:

– на рабочем столе щелкните правой кнопкой значок папки **Корзина** и выберите команду **Свойства**;

– перетащите бегунок регулятора, задающего объем папки **Корзина**.

Примечания

1 Если требуется задать различную настройку для разных дисков, в диалоговом окне **Свойства: Корзина** выберите параметр **Независимая**

**конфигурация дисков**, а затем выберите вкладку диска, для которого требуется изменить настройку.

2 Если требуется использовать общую настройку для всех дисков, выберите параметр **Единые параметры для всех дисков**.

**4.2.2.5** Файлы, помещенные в корзину, находятся в папке **Recycled** корневого каталога каждого диска (обычно она скрыта). При работе с корзиной возможны следующие операции:

– если удалить или переименовать папку **Recycled,** она будет создана заново при запуске Windows 9x и, очевидно, будет пустой;

– если присвоить папке **Корзина** первоначальное имя, все ранее удаленные файлы появятся в корзине снова.

**4.2.2.6** Хранящийся в папке **Корзина** файл занимает столько же места на диске, сколько занимал перед тем, как его уничтожили. Если вы твердо уверены в том, что тот или иной файл больше вам никогда не понадобится, можете удалить его обычным способом, а затем убрать из папки **Корзина**. Для удаления файла из папки **Корзина** откройте ее, выберите файл и нажмите клавишу **Del** – отмеченный объект будет удален навсегда.

Для очистка папки **Корзина** (т.е. удаления всего ее содержимого) щелкните по ее значку правой кнопкой и выберите в ее меню команду **Очистить корзину** или же, если папка **Корзина** открыта, выберите эту команду в меню **Файл**.

Примечания

1 Для удаления из папки **Корзина** группы объектов, удерживая нажатой клавишу **Ctrl**, щелкните по каждому объекту группы. Если же необходимые объекты располагаются в окне один за другим, щелкните по первому, а затем – по последнему, удерживая нажатой клавишу **Shift**. Далее нажмите клавишу **Del**.

2 Для открытия файла, находящегося в папке **Корзина**, перетащите его значок на рабочий стол, а затем дважды щелкните по значку.

**4.2.3** Утилита **MSBackup**

**4.2.3.1** Утилита **MSBackup** осуществляет резервное копирование файлов (т.е. осуществляет архивацию) и восстанавливает оригиналы в случае их разрушения или переноса на другую ЭВМ. Позволяет создавать архивы в сжатом и обычном состоянии, причем в своем собственном формате, что способствует защите информации. Отметим, что степень сжатия здесь выше, чем при работе программы **DriveSpace**, поскольку здесь не предъявляются жесткие требования к быстродействию.

Процесс архивации (резервного копирования) включает три стадии:

а) выбор архивируемых файлов и папок путем установки флажков, расположенных слева от их имен;

б) выбор устройства (например, диска **A:**), куда будут помещены копии файлов;

в) собственно резервное копирование файлов.

Архиватор позволяет:

– создать полный архив системы;

– архивировать выбранные папки и файлы;

– восстановить систему;
– восстановить файлы и папки из архива;
– сравнить содержимое архива с оригиналом;
– исключать файлы по дате и времени создания;
– устанавливать параметры архивации;
– работать с лентами.
– создать архив переносом папок;
– определить задание архивации;
– архивировать готовое задание архивации.

**4.2.3.2** Для запуска программы архивирования следует выбрать **Пуск – Программы – Стандартные – Служебные программы – Архивация данных (Backup)** или с помощью главного меню Windows выполнить **Пуск – Выполнить… – msbackup – OK**.

**4.2.3.3** Для создания полной резервной копии системы (резервирования всего содержимого компьютера) следует сделать следующее:

– в меню **Задание** выберите команду **Создать**;

– разверните папку **Мой компьютер** и установите флажок рядом с каждым диском;

– нажмите кнопку **Параметры** и выберите вкладку **Дополнительно** в диалоговом окне **Параметры задания архивации**;

– установите флажок **Архивировать реестр Windows** и нажмите кнопку **OK**;

– нажмите кнопку **Запуск**.

Примечания

1 Настоятельно рекомендуется регулярно выполнять полное резервирование системы. Это позволит иметь текущий системный архив для восстановления в случае сбоя жесткого диска.

2 Сетевые диски могут быть выбраны для архивации аналогично любому обычному диску.

3 Полную резервную копию системы можно создать также путем перетаскивания значка полной резервной копии системы на значок программы **Backup**.

**4.2.3.4** Для создания резервной копии выбранных папок и файлов (задания архивации) следует выбрать вкладку **Архивация данных** и далее:

– в меню **Задание** выбрать команду **Создать**. Открывается окно **Backup**, в котором нет выбранных файлов, а все параметры имеют значения по умолчанию;

– выбрать параметр **Все выбранные файлы** или **Новые и измененные файлы**;

– установить флажки для дисков, папок и файлов, которые следует архивировать. Выбор файлов необходимо выполнить до запуска задания архивации;

– выбрать место назначения архива в поле со списком **Архивировать файлы в**;

– нажмите кнопку **Параметры** для установки параметров задания;

– выбрать вкладки и просмотреть доступные группы параметров. Параметры размещаются на следующих вкладках: **Общие, Пароль, Тип, Исключение, Отчет, Дополнительно**;

– в меню **Задание** выбрать команду **Сохранить**;

– для запуска задания архивации нажать кнопку **Запуск**.

В следующий раз, когда потребуется создать архив тех же файлов, можно использовать уже имеющееся задание архивации.

Примечания

1 Синие метки показывают, что диски, папки или файлы выбраны для архивирования. Серая метка рядом со значком диска или папки показывает, что для архивирования выбраны некоторые, но не все файлы на диске или в папке.

2 Сетевые диски могут быть выбраны для архивации аналогично любому обычному диску.

**4.2.3.5** Для сохранения нового задания архивации необходимо:

– в меню **Задание** выбрать команду **Сохранить как**;

– ввести имя задания архивации в поле **Имя задания**;

– нажать кнопку **Сохранить**.

Примечание – Имя задания архивации может содержать до 130 символов, включая пробелы.

*Задание архивации* (набор папок и файлов) представляет собой специальный файл, содержащий список файлов и каталогов, включаемых в архив. Набор файлов весьма удобен в том случае, если часто осуществляется резервное копирование нескольких одних и тех же файлов, поскольку позволяет несколько раз использовать ранее уже определенный состав архива.

**4.2.3.6** Для изменения задания архивации необходимо:

– в окне **Backup** выбрать в поле со списком **Задание архивации** задание, которое требуется изменить;

– изменить набор отобранных дисков, папок или файлов, а также другие настройки;

– в меню **Задание** выбрать команду **Сохранить**.

Примечания

1 Имеется возможность выбрать элементы для архивации или отказаться от выбора. Кроме того, пользователь имеет возможность изменить параметры в диалоговом окне **Параметры задания архивации**. Чтобы открыть диалоговое окно **Параметры задания архивации**, нажмите кнопку **Параметры**.

2 Если требуется сохранить измененное задание архивации под новым именем, выберите в меню **Задание** команду **Сохранить как**.

Для создания резервной копии на основе имеющегося задания архивации следует в меню **Задание** выбрать команду **Открыть…**, выбрать нужное задание, нажать кнопку **Открыть** (или сразу в меню **Задание** выбрать нужное задание) и

затем, после выполнения определенных действий (см. подпункт 4.2.3.4), запустить на архивацию.

**4.2.3.7** Для выбора типа архива необходимо:

– установить в окне **Backup** переключатель **Все выбранные файлы** или **Новые и измененные файлы**;

– для изменения типа архива для новых и измененных файлов в области **Выбор способа архивации** нажмите кнопку **Параметры**;

– выбрать вкладку **Тип**;

– выбрать параметр **Только новые и измененные файлы**;

– выбрать параметр типа архива **Разностный** или **Добавочный**;

– нажать кнопку **OK**.

Примечания

1 В *разностный тип* архива помещаются те из выбранных файлов, которые были изменены после последней архивации всех файлов.

2 В *добавочный тип* архива помещаются все выбранные файлы, которые были изменены после последней архивации всех файлов или добавочной архивации.

**4.2.3.8** Для выполнения операции восстановления необходимо:

– выбрать вкладку **Восстановление**;

– выбрать устройство в поле со списком **Выбор архива**;

– установить флажки для дисков, папок и файлов, которые следует восстанавливать;

– выбрать место восстановления в поле со списком **Выбор места восстановления файлов**;

– нажать кнопку **Параметры** для установки параметров задания;

– выбрать один из следующих параметров: **Не заменять**, **Заменить старые версии файлов** или **Всегда заменять –** и нажать кнопку **OK**;

– нажать кнопку **Запуск**.

**4.2.3.9** При работе с утилитой **Msbackup** удобно пользоваться мастерами соответственно архивации и восстановления файлов. Доступ к ним можно получить из окна **Backup** командами **Сервис – Мастер архивации файлов…** и **Сервис – Мастер восстановления файлов…**.

**4.2.3.10** На быстродействие программы архивации оказывают влияние следующие факторы:

а) *сжатие данных* – позволяет уменьшить общее время архивации и требуемый для сохранения объем. Чтобы включить сжатие, нажмите кнопку **Параметры** на вкладке **Архивация данных** и выберите параметр **сжимать данные**. Если данные уже были сжаты, выберите параметр **никогда не сжимать данные**. Попытка сжатия данных, которые уже были сжаты, приведет к тому, что данные будут занимать больше места на носителе;

б) *сравнение данных* – для проверки данные считываются с носителя архива и сравниваются с данными в буфере (памяти). Выполнение архивации с этим параметром может привести к удвоению времени архивирования. Для повышения скорости снимайте флажок **Сравнивать файлы**;

в) *архивация больших наборов файлов* (десятки или сотни тысяч файлов) требует большого объема памяти. Программа архивации не ограничивает число папок или файлов, которые могут содержаться в одном архиве. Однако при выполнении архивирования на компьютере с небольшим объемом памяти полезно разбивать большие архивы на несколько частей;

г) *использование памяти* – программа архивации должна обеспечивать возможность использования других приложений во время создания или восстановления архива. Однако в фоновом режиме архивы создаются медленнее и могут потребовать больше места на носителе. Если скорость является существенным фактором, следует закрыть все другие приложения.

**4.2.4** Программы-архиваторы

Существует много программ–архиваторов. Пожалуй, наибольшее распространение из них получили **PKZIP**, **ARJ** и **RAR**. Как правило, они сжимают файл сильней, чем **MSBackup**, но последний является “родным” для Windows, что и определяет его удобство.

**4.2.5** Атрибуты папок, файлов и ярлыков

**4.2.5.1** *Атрибуты* – это пометки, которые используются в системе для обозначения некоторых характеристик файлов. В файловой системе Windows папки, файлы и ярлыки могут быть без атрибутов или иметь любую комбинацию следующих пометок: **Архивный**, **Скрытый**, **Системный** и **Только для чтения**. Атрибуты указанных объектов предоставляют дополнительные возможности по защите данных. В списке свойств вы можете узнать атрибуты конкретных объектов и при необходимости изменить их.

**4.2.5.2** Атрибут **Архивный** указывает на то, что в файл были внесены изменения за время, прошедшее с момента выполнения последней операции архивации (резервного копирования). Каждый раз, когда вы создаете новый файл или изменяете содержимое уже существующего, Windows устанавливает для него этот атрибут. Архиваторы обычно обнуляют его в процессе архивации, в частности, это делает **MSBackup**. Если же вы внесете изменения в файл после его архивации, то атрибут **Архивный** выставляется вновь – для того, чтобы ваша программа-архиватор (программа резервного копирования) распознала этот файл как требующий повторного сохранения в архиве.

**4.2.5.3** Некоторые программы используют атрибуты **Скрытый** и **Системный**, чтобы пометить важные файлы, которые запрещено удалять и изменять ввиду того, что они необходимы для корректной работы приложений Windows. При попытке их удаления Windows выведет дополнительное окно для подтверждения удаления.

**4.2.5.4** Файл с атрибутом **Только для чтения** можно открыть, но сохранить под тем же именем невозможно. Некоторые программы устанавливают этот атрибут, чтобы предотвратить случайное изменение файлов. Очень часто атрибут **Только для чтения** предотвращает не только изменение, но и удаление объекта. Так, например, команды **erase** и **del** операционной системы MS DOS не уничтожают файлы с атрибутом **Только для чтения** – на экран в этом случае будет выведено сообщение **Доступ запрещен**.

**4.2.5.5** Для изменения свойств файла или папки необходимо:

– в окне **Мой компьютер** или в окне проводника выбрать файл или папку, свойства которой требуется изменить;

– в меню **Файл** выбрать команду **Свойства**;

– выполнить необходимые действия в диалоговом окне **Свойства**.

Примечания

1 Можно также щелкнуть по папке или файлу правой кнопкой и выбрать команду **Свойства**.

2 Для выделения папки в левой области окна проводника щелкните по значку папки.

## 4.3 Порядок выполнения работы

**4.3.1** Все нижеперечисленные действия должны быть отражены в демонстрационном текстовом файле отчета соответствующими окнами (экранами) и текстом.

**4.3.2** Получить справку о командах DOS, перечисленных в пункте 4.2.1.

**4.3.3** Создать на диске D: (или E:) папку A, а в ней B. Скопировать в папку A папку C:\Мои документы, а в папку B: — любую папку на диске D: (или E:). Зафиксировать объем папки A.

**4.3.4** Создать полный архив папки A с помощью **MSBackup**. Зафиксировать объем архива.

**4.3.5** Удалить папку A.

**4.3.6** Восстановить папку A из архива.

**4.3.7** Выбрать несколько файлов, создать задание архивации с именем **Задание** и заархивировать эту группу файлов в архив с именем **Архив**.

**4.3.8** Удалить или изменить эти файлы, вошедшие в **Задание**.

**4.3.9** Восстановить файлы из архива с именем **Архив**.

**4.3.10** Создать на диске D: (или E:) папку C.

**4.3.11** Восстановить архив папки A в папку C.

**4.3.12** Заархивировать папку B на дискеты.

**4.3.13** Удалить папку B.

**4.3.14** Восстановить папку B с дискет.

**4.3.15** Выполнить пункт 4.3.4 архиваторами **ZIP**, **ARJ**, **LHA**, **RAR**, **UC2** и **ACE** (доступ к этим архиваторам можно получить через команду **Файл – Упаковать…** программы Windows Commander). Результаты архивации представить в виде таблицы. Сделать вывод.

**4.3.16** После демонстрации файла отчета преподавателю удалить файл отчета и все созданные архивы и папки.

## 4.4 Контрольные вопросы

**4.4.1** Перечислить основные команды копирования-восстановления.

**4.4.2** Сколько места занимает корзина?

**4.4.3** Как задать дисковое пространство для корзины?

**4.4.4** Что произойдет, если изменить название корзины на одном из локальных дисков?

**4.4.5** Как восстановить удаленную папку?

**4.4.6** Как восстановить удаленный файл в другую папку?

**4.4.7** Перечислите три стадии процесса архивации.

**4.4.8** Что позволяет выполнить архиватор **MSBackup**?

**4.4.9** Как запустить архиватор **MSBackup**?

**4.4.10** Пояснить на экране органы управления архиватором.

**4.4.11** Что такое задание архивации? Где его можно просмотреть? Каким образом его можно изменить?

**4.4.12** Как поместить в архив только обновленные файлы?

**4.4.13** От каких основных факторов зависит быстродействие программы архивации?

**4.4.14** Назовите атрибуты файлов (папок, ярлыков). На что они могут влиять и как их можно установить (изменить)?

# 5 МОДЕЛИ ОПЕРАЦИОННЫХ СИСТЕМ. СЕТИ ОЧЕРЕДЕЙ ОЖИДАНИЯ (СЕТЬ ДЖЕКСОНА)

## 5.1 Цель работы

**5.1.1** Получить сведения о модели операционной системы типа сети очередей ожидания (сети Джексона).

**5.1.2** Изучить влияние входных и внутренних параметров конкретной открытой сети Джексона на выходные параметры ВС с помощью имитационной и аналитической моделей.

## 5.2 Описание сети Джексона

**5.2.1** Введение

Задача организации эффективного совместного использования ресурсов несколькими процессами является весьма сложной, и сложность эта порождается в основном случайным характером возникновения запросов на потребление ресурсов. В мультипрограммной системе образуются очереди заявок от одновременно выполняемых программ к разделяемым ресурсам компьютера: процессору, странице памяти, принтеру, диску. Операционная система организует обслуживание этих очередей по разным алгоритмам: в порядке поступления, на основе приоритетов, кругового обслуживания и т.д. Анализ и определение оптимальных дисциплин обслуживания заявок являются предметом специальной области прикладной математики – теории массового обслуживания. Очень часто в ОС реализуются и эмпирические алгоритмы обслуживания очередей, прошедшие проверку практикой.

**5.2.2** Модели с одной очередью ожидания

Модель, используемая для распределения одного ресурса, является моделью типа "обслуживающее устройство с очередью ожидания", схема которой представлена на рисунке 5.1 [10]. Пользователи (клиенты) запрашивают некоторую

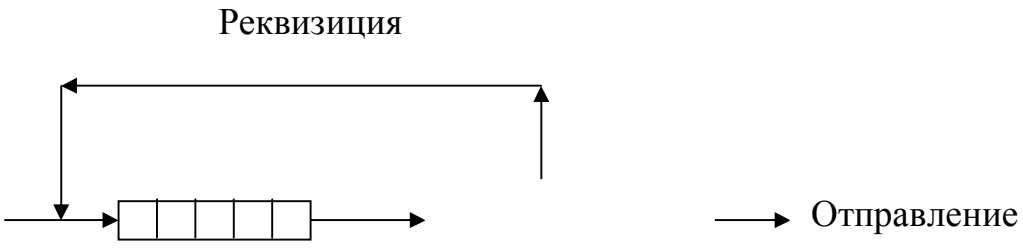



Рисунок 5.1 – Обслуживающее устройство с очередью

услугу, соответствующую исключительному использованию обслуживающего устройства в течение некоторого времени (времени обслуживания). Устройство может обслуживать за один раз только одного пользователя; другие пользователи, поступающие во время обслуживания (устройство занято), ожидают в очереди. Таким образом, устройство может рассматриваться как ресурс с исключительным доступом, а время обслуживания - как длительность использования ресурса, и, следовательно, результаты изучения простых очередей ожидания непосредственно применимы к задаче распределения одного ресурса. Реквизиция означает вынужденное прекращение использования ресурса по команде распределителя ресурсов (распределитель может быть процессом, запросы к которому передаются с помощью сообщений, или же монитором, специальная процедура которого служит для передачи запросов).

Времена прибытия клиентов и сроки обслуживания, которые потребуются, заранее не известны: это случайные переменные с некоторым законом распределения.

Итак, модель определяется тремя характеристиками:

– распределением моментов прибытия клиентов;

– распределением запрашиваемых сроков обслуживания;

– способом управления очередью ожидания.

Ограничимся простейшей моделью для распределения моментов прибытия и времён обслуживания - законом Пуассона, с дисциплиной выбора из очереди FIFO (FCFS) "первым пришёл, первым обслужен" ("первый пришедший обслуживается первым"), тогда по Кендаллу модель на рисунке 5.1 можно обозначить как M/M/1 [1, 6, 10]. Эта модель проста с точки зрения вычислений и в то же время достаточно адекватно описывает некоторые реальные ситуации. Она позволяет выделить основные параметры распределения и получить (по крайней мере, качественно) представление об их влиянии. Модель простой очереди ожидания полезна для локального представления поведения части системы или когда система в целом отождествляется с одним обслуживающим устройством.

**5.2.3** Сети очередей ожидания

Для представления сложной системы используют сети очередей ожидания, которые включают несколько обслуживающих устройств; пользователь следует по одному из путей сети, последовательно обращаясь к разным обслуживающим устройствам. Общим методом, используемым для анализа работы с сетями очередей ожидания, является имитация; однако могут быть использованы и аналитические модели с учётом предположений, которые мы уточним в дальнейшем.

Сеть Джексона определяется следующими свойствами [10]:

– каждый из N узлов сети является либо одним обслуживающим устройством, либо совокупностью идентичных устройств, работающих параллельно; законы обслуживания являются экспоненциальными;

– после обслуживания в узле i пользователь (клиент, требование, процесс) может запросить обслуживание в узле j или покинуть сеть; вероятности переходов фиксированы;

– в случае открытой сети снабжение осуществляется извне с помощью нескольких независимых источников клиентов, каждый из которых генерирует пуассоновский поток; для закрытой сети число клиентов фиксировано и нет ни входов, ни выходов.

В случае открытой сети можно показать, что каждое обслуживающее устройство ведёт себя так, как при обслуживании пуассоновского входного потока (хотя в действительности это не так). Тогда достаточно вычислить распределение средних потоков в сети, пользуясь фиксированными вероятностями перехода и записывая уравнения сохранения потока:

$$\Lambda_i = \lambda_i + \sum_{j=1}^{N} p_{ji} \cdot \Lambda_j, \quad i \neq j, \quad i = 1,...,N. \tag{5.1}$$

Условие существования стационарного состояния:

$$\Lambda_i / \mu_i < 1, \quad i = 1,...,N, \tag{5.2}$$

где $\mu_i$ - интенсивность обслуживания в i-м устройстве.

**5.2.4** Система с несколькими обслуживающими устройствами

Рассмотрим конкретную открытую сеть Джексона, схематически представленную на рисунке 5.2. Задания поступают от n независимо работающих

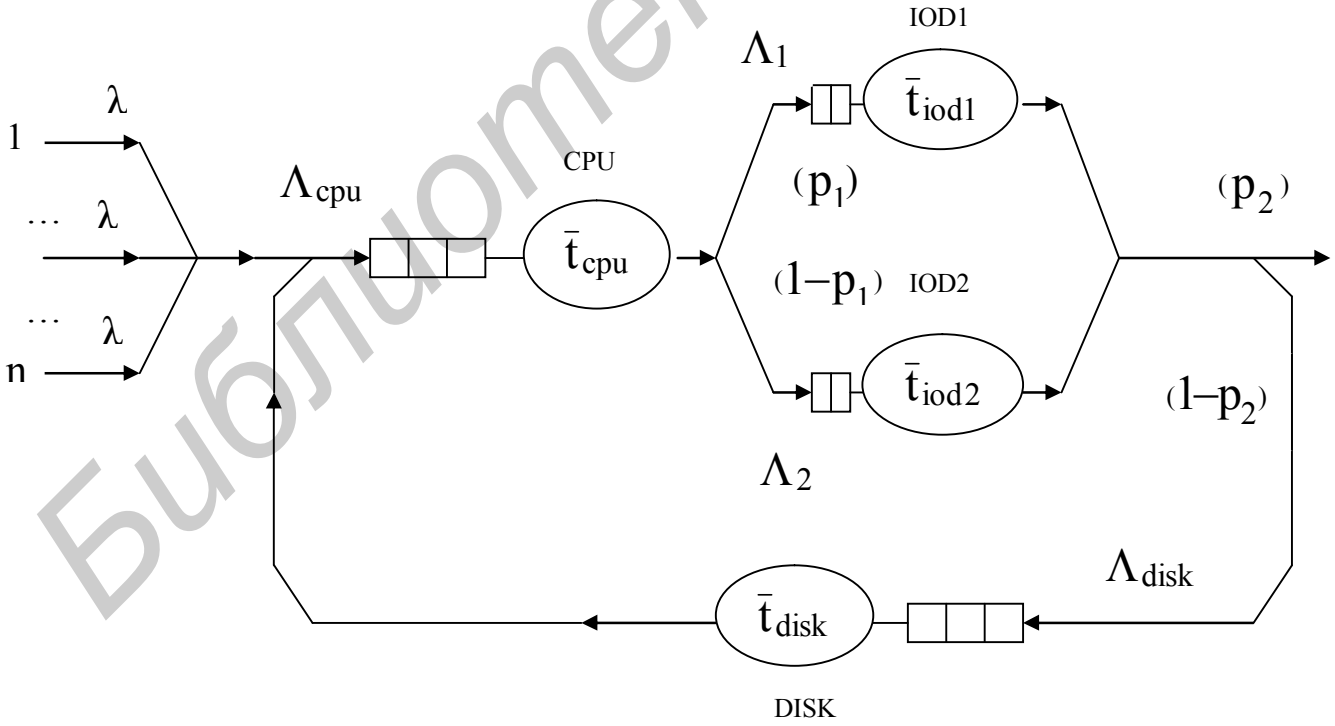



Рисунок 5.2 – Пример сети очередей ожидания

терминалов, в среднем идентичных друг другу. Подразумевается, что в мультипрограммной ВС реализован невытесняющий алгоритм планирования с дисциплиной FIFO. Для каждого задания требуются процессор CPU и устройство ввода-вывода (УВВ), выбираемое из IOD1 и IOD2 с вероятностями $p_1$ и $1-p_1$ соответственно. После этого можно либо закончить работу с вероятностью $p_2$, либо запросить внешний диск DISK прочитать файл начать цикл обработки задания снова. Мы предполагаем, что законы обслуживания устройств CPU, IOD1, IOD2 и DISK экспоненциальные (средние величины $\bar{t}_{cpu}$, $\bar{t}_{iod1}$, $\bar{t}_{iod2}$, и $\bar{t}_{disk}$), вероятности перехода $p_1$ и $p_2$ фиксированы и запросы от каждого терминала распределены по закону Пуассона с параметром $\lambda$. Следовательно, можно воспользоваться моделью Джексона. Уравнения сохранения потока [см.(5.1)] записываются в следующем виде:

$$\left.\begin{aligned} &\Lambda_{cpu} = n\cdot\lambda + \Lambda_{disk}, \qquad \Lambda_1 = p_1\cdot\Lambda_{cpu}, \\ &\Lambda_{disk} = (1-p_2)\cdot\Lambda_{cpu}, \quad \Lambda_2 = (1-p_1)\cdot\Lambda_{cpu}. \end{aligned}\right\} \tag{5.3}$$

Из системы (5.3) получаем потоки для каждого обслуживающего устройства, относящегося к типу M/M/1:

$$\left.\begin{aligned} &\Lambda_{cpu} = \frac{n\cdot\lambda}{p_2}, \qquad \Lambda_1 = \frac{p_1\cdot n\cdot\lambda}{p_2}, \\ &\Lambda_2 = \frac{(1-p_1)\cdot n\cdot\lambda}{p_2}, \quad \Lambda_{disk} = \frac{(1-p_2)\cdot n\cdot\lambda}{p_2}. \end{aligned}\right\} \tag{5.4}$$

Остальные параметры модели рассчитываются по следующим формулам:

– коэффициент загрузки i-го устройства

$$K_i = \Lambda_i\cdot\bar{t}_i; \tag{5.5}$$

– среднее число заявок в очереди $Q_i$ к i-му устройству

$$\overline{N}_{Qi} = \frac{K_i^2}{1-K_i}; \tag{5.6}$$

– среднее время ожидания заявки в очереди $Q_i$ к i-му устройству

$$\overline{W}_{Qi} = \frac{\overline{N}_{Qi}}{\Lambda_i} = \frac{K_i\cdot\bar{t}_i}{1-K_i}. \tag{5.7}$$

**5.2.5** Применение моделей с очередями ожидания

Модель с очередями ожидания является в основном средством прогнозирования, позволяющим:

– оценить при проектировании глобальные характеристики ВС;

– определить критические параметры и узкие места системы;

– оценить последствия модификаций системы или условий её использования.

Для этого могут использоваться как математические, так и имитационные модели. По сравнению с аналитическим подходом преимущество имитации заключается в том, что не ограничивается сложность используемой модели. Другое преимущество состоит в том, что с помощью имитации можно получить результаты для переходных процессов и стационарных режимов, в то время как аналитические (математические) модели описывают обычно (в случае моделирования СМО) установившиеся режимы.

## 5.3 Исходные данные для аналитического расчета модели

Число терминалов n зависит от номера варианта (таблица 5.1). Остальные данные следующие:

$\lambda = 0{,}01\ \text{s}^{-1}$; $\bar{t}_{cpu} = 0{,}8$ s; $\bar{t}_{iod1} = 0{,}3$ s; $\bar{t}_{iod2} = 0{,}6$ s; $\bar{t}_{disk} = 1$ s; $p_1 = 0{,}6$; $p_2 = 0{,}4$.

Таблица 5.1

| Число терми-налов | Номер варианта | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| n | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 52 | 56 | 58 | 60 | 65 | 70 | 75 | 80 |

## 5.4 Порядок выполнения работы

**5.4.1** Получить у преподавателя номер варианта исходных данных для модели, представленной на рисунке 5.2.

**5.4.2** Скопировать в свой рабочий каталог D:\OS из каталога D:\OS\LABOS\LABOS5 файлы modos5.gps (содержащий исходный текст имитационной модели на языке GPSS/PC), startup.gps и lab5.bat. Предварительно в рабочем каталоге должны быть размещены следующие обязательные файлы: gpsspc.exe, gpssrept.exe, settings.gps и position.gps (из каталога D:\OS\GPSS).

**5.4.3** Изменить содержание поля операндов оператора (из файла modos5.gps)

VARIANT VARIABLE номер_варианта

в соответствии со своим вариантом.

**5.4.4** Выяснить, имеет ли система стационарное состояние [см. (5.2)], выявить узкие места системы и, в случае необходимости, определить необходимое увеличение возможностей системы для заданных внешних условий функционирования. Изменять возможности системы следует через изменение

средних времён обработки заявки на устройствах сети. Это делается через изменение поля операндов следующих операторов (из файла modos5.gps):

- для CPU: T_CPU VARIABLE время обработки в миллисекундах ;
- для IOD1: T_IOD1 VARIABLE -"- ;
- для IOD2: T_IOD2 VARIABLE -"- ;
- для DISK: T_DISK VARIABLE -"- .

**5.4.5** Получить основные характеристики системы с помощью аналитической модели.

**5.4.6** Получить результаты моделирования с помощью имитационной модели, запустив модель на счет с помощью файла lab5.bat; далее см. файл статистики modos5.txt.

Примечания

1 Коэффициенты загрузок устройств даны в файле modos5.txt в долях от тысячи.

2 Уточненные значения средних длин очередей (с точностью до двух знаков после запятой) см. в файле modos5.txt в колонке AVE.CONT. объектов типа QUEUE (очередь).

**5.4.7** Сравнить результаты имитационной и аналитической моделей.

**5.4.8** Проверить с помощью имитационного моделирования, будет ли в системе стационарное состояние при соблюдении для процессора условия $\Lambda_i / \mu_i = 1$. Сделать вывод.

**5.4.9** Оформить отчет и сформулировать общие выводы по работе (в виде текстового файла).

## 5.5 Контрольные вопросы

**5.5.1** Дать краткое описание модели с одной очередью ожидания.

**5.5.2** Что представляют собой сети Джексона, какие они бывают?

**5.5.3** Сформулировать условие существования стационарного состояния ВС.

**5.5.4** Что означает термин "узкое место в системе"?

**5.5.5** Каким образом можно устранить узкие места в системе?

**5.5.6** Перечислите основные выходные характеристики качества функционирования ВС.

**5.5.7** Уметь составлять аналитическое описание сети Джексона заданной структуры и анализировать ее.

# 6 МОДЕЛИ ОПЕРАЦИОННЫХ СИСТЕМ. СИСТЕМА С РАЗДЕЛЕНИЕМ ВРЕМЕНИ

## 6.1 Цель работы

**6.1.1** Получить сведения о модели операционной системы, реализующей режим разделения времени.

**6.1.2** Изучить влияние внутренних параметров конкретной модели системы разделения времени при разных дисциплинах планирования работы центрального процессора на выходные параметры ВС с помощью имитационных GPSS-моделей.

## 6.2 Описание системы с разделением времени

**6.2.1** Понятие режима разделения времени

**6.2.1.1** Понятие разделения времени относится к системам, основная задача которых состоит в одновременном предоставлении вычислительных средств и некоторых услуг пользователям, находящимся за терминалами. При этом предполагается, что у каждого из них создаётся полная иллюзия монопольной работы с вычислительной системой. В качестве пользователей могут выступать как разработчики новых программных средств, так и конечные пользователи, заинтересованные в получении результатов выполнения определённых программ [1, 5, 8, 10, 11,13].

**6.2.1.2** Функционирование системы разделения времени (СРВ) главным образом имеет целью оперативное обслуживание пользователей (или пользователя, работающего одновременно с несколькими приложениями) и поддержание с ними активного диалога. Эта цель иногда достигается за счёт некоторого снижения эффективности использования оборудования. Несмотря на то что пакетные системы с мультипрограммированием и системы разделения времени имеют немало аналогичных механизмов, обеспечивающих одновременную работу и взаимодействие устройств ВС, назначения этих систем (критерии их эффективности) совершенно различны. Пакетные системы с мультипрограммированием обеспечивают наиболее эффективное использование аппаратуры и максимальную пропускную способность. Системы же разделения времени предназначаются для оперативного обслуживания запросов пользователей, т.е. для удобства их работы.

**6.2.1.3** Системы разделения времени призваны исправить основной недостаток систем пакетной обработки – изоляцию пользователя-программиста от процесса выполнения его задач. Каждому пользователю в этом случае предоставляется терминал, с которого он может вести диалог со своей программой. Так как в системе разделения времени каждой задаче выделяется только квант

процессорного времени, ни одна задача не занимает процессор надолго и время ответа оказывается приемлемым. Если квант выбрать достаточно небольшим, то у всех пользователей, одновременно работающих на одной и той же машине, складывается впечатление, что каждый из них единолично использует машину.

**6.2.2** Управление процессором

**6.2.2.1** Процессы получают возможность выполнять конкретную работу, когда в их распоряжение выделяются физические процессоры. Говорят, что процесс работает (выполняется, развивается, является активным или избранным), если в данный момент в его распоряжение предоставлен центральный процессор (ЦП).

Распределение процессоров по процессам представляет собой сложную задачу, которую решает ОС. Определение того, когда следует выделять процессоры и каким именно процессам, называется планированием загрузки процессоров (планированием процессов).

В СРВ реализуются вытесняющие алгоритмы планирования, основанные на концепции *квантования*. В соответствии с этой концепцией каждому процессу поочередно для выполнения выделяется ограниченный непрерывный период процессорного времени – *квант*. Смена активного процесса происходит, если:

– процесс завершился и покинул систему;
– произошла ошибка;
– процесс перешел в состояние ожидания (блокировки);
– истек квант процессорного времени, отведенный данному процессу.

**6.2.2.2** При планировании могут приниматься во внимание приоритет процессов, время их ожидания в очереди, накопленное время выполнения, интенсивность обращения к вводу-выводу и другие факторы.

*Приоритет* процесса – это число, характеризующее степень привилегированности процесса при использовании ресурсов ВС, в частности процессорного времени: чем выше приоритет, тем выше привилегии, тем меньше времени будет проводить процесс в очередях. Приоритет может выражаться целым или дробным, положительным или отрицательным значением. В некоторых ОС принято, что приоритет процесса тем выше, чем больше (в арифметическом смысле) число, обозначающее приоритет. В других системах, наоборот, чем меньше число, тем выше приоритет.

Приоритет процесса назначается операционной системой при его создании (значение приоритета включается в описатель процесса), при этом ОС учитывает, является ли этот процесс системным или прикладным, каков статус пользователя, запустившего процесс, было ли явное указание пользователя на присвоение процессу определенного уровня приоритета. Различают следующие приоритеты [1, 5, 8, 10, 11, 13]:

– *статический* (*фиксированный*) приоритет, определённый заранее, например, как функция запрашиваемого времени обслуживания и т.п.; статические приоритеты не изменяются и в силу этого не реагируют на изменения в окружающей ситуации, которые (изменения) могут сделать желательной корректировку приоритетов;

– *динамический* приоритет, т.е. изменяющийся с течением времени приоритет, например, как функция прошедшего времени ожидания, прошедшего времени обслуживания и т.п.; здесь начальное значение приоритета, присвоенное процессу, может действовать в течение лишь короткого периода времени, после чего назначается новое, более подходящее значение; механизмы динамических приоритетов способствуют повышению реактивности системы;

– *относительный* приоритет – в системах с относительными приоритетами активный процесс не может быть прерван другим процессом, появившимся в очереди готовности, даже если он обладает более высоким уровнем приоритета;

– *абсолютный* приоритет – в системах с абсолютными приоритетами выполнение активного процесса прерывается, кроме указанных выше причин (см. подпункт 6.2.2.1), еще при одном условии: если в очереди готовых процессов появился процесс, приоритет которого выше приоритета активного потока. В этом случае прерванный процесс переходит в состояние готовности.

Таким образом, система присваивает приоритеты автоматически, при этом они могут присваиваться по какому-то рациональному принципу или произвольно назначаться в ситуациях, когда системному механизму необходимо каким-то образом различать процессы, однако он не знает, какой из них в действительности более важен.

**6.2.3** Планирование с переключением (с перераспределением) и без переключения (без перераспределения)

**6.2.3.1** Если после предоставления ЦП в распоряжение некоторого процесса отобрать «силой» ЦП у этого процесса нельзя, то в этом случае говорят о невытесняющем алгоритме планирования, т.е. о дисциплине планирования без переключения (без перераспределения или без реквизиции). Если же ЦП можно отобрать, то говорят о вытесняющем алгоритме планирования, или о дисциплине планирования с переключением (с реквизицией). В системах разделения времени используется, естественно, дисциплина планирования с переключением.

**6.2.3.2** Полезной компонентой системы планирования с переключением является *интервальный таймер*, или прерывающие часы. По истечении заданного временного интервала таймер генерирует сигнал прерывания, по которому управление центральным процессором переходит от текущего процесса к операционной системе. Операционная система может после этого запустить в работу следующий процесс.

**6.2.4** Квантование времени

**6.2.4.1** В простейшей системе разделения времени, где каждому терминалу поставлена в соответствие некоторая фиксированная рабочая область памяти и где непосредственно с терминала нельзя заказать выполнение операции ввода-вывода, распределение процессорного времени между отдельными пользователями является исключительным правом ОС. Если при этом все задания более или менее однородны и требуют примерно одинакового количества ресурсов, а каждый пользователь запрашивает выполнение примерно одинаковых сервисных функций, то работой системы вполне может управлять простой планировщик, реализующий режим квантования времени. Квантование – это фактически выделение для выполнения

каждой программы установленного временного интервала – кванта q. По завершении этого интервала данная программа приостанавливается и начинает выполняться следующая. Планировщик циклически просматривает свою очередь, предоставляя всем программам по очереди фиксированные интервалы обслуживания.

**6.2.4.2** Конечно, квантование может быть и более изощрённым. В частности, продолжительность интервала обслуживания каждой программы может рассчитываться на основе её приоритета. Таким образом, программе со значением приоритета, равным трём, вполне естественно предоставить тройной интервал обслуживания; программе со значением приоритета, равным шести, – интервал обслуживания в шесть раз больший, чем минимальный, и т.д. Можно также представить себе алгоритм планирования, в котором первоначально каждому процессу назначается достаточно большой квант, а величина каждого следующего кванта уменьшается до некоторой заданной величины (преимущество получают короткие задачи) или, наоборот, каждый следующий квант, выделяемый определенному процессу, больше предыдущего (уменьшаются накладные расходы на переключение задач, когда одновременно выполняется несколько длинных задач).

**6.2.5** Размер кванта времени

**6.2.5.1** Определение размера кванта времени имеет критическое значение для эффективной работы ВС. Следует ли выбирать большой или малый квант времени? Следует ли делать его фиксированным или переменным? Следует ли задавать одно и то же значение кванта времени для всех пользователей или нужно определять его индивидуально для каждого пользователя?

**6.2.5.2** Если квант времени становится очень большим, то каждому процессу предоставляется практически столько времени, сколько ему требуется для завершения, так что циклическое планирование, по сути, вырождается в планирование по принципу FIFO, когда процесс, получивший ЦП в своё распоряжение, выполняется до конца (т.е. принцип FIFO - это дисциплина планирования без переключения). Если квант времени становится очень маленьким, то накладные расходы на переключения (при переключении ЦП захватывает ОС) начинают играть доминирующую роль, причём характеристики системы, в конце концов, настолько ухудшаются, что с какого-то момента основное время затрачивается на переключение процессора, так что лишь незначительная часть времени остаётся (если вообще остаётся) на выполнение вычислений для пользователей.

**6.2.5.3** Рассмотрим предположительно оптимальное значение кванта времени (небольшую долю секунды), при котором обеспечиваются хорошие времена ответа. Подобный квант времени q достаточно велик, так как подавляющее большинство интерактивных запросов требует для своего обслуживания меньшего времени, чем длительность кванта. Когда интерактивный процесс начинает выполняться, то он, как правило, использует ЦП в течение некоторого времени, после чего генерирует запрос ввода-вывода. Когда запрос ввода-вывода выдан, этот процесс уступает ЦП следующему процессу. Поскольку величина кванта больше, чем это время вычислений до формирования запроса ввода-вывода, процессы пользователей

выполняются практически с максимальной скоростью. Каждый раз, когда процесс пользователя получает в своё распоряжение ЦП, он с большой вероятностью доработает до момента выдачи запроса ввода-вывода. Благодаря этому сводятся к минимуму накладные расходы на диспетчирование, обеспечиваются максимальное использование ресурсов ввода-вывода и относительно короткие времена ответа.

**6.2.5.4** Так какова же величина подобного оптимального кванта времени q? Очевидно, что она меняется от системы к системе, причём меняется в зависимости и от нагрузок. Если все процессы лимитируются ЦП, то вообще не имеет смысла переключаться с процесса на процесс. Это объясняется тем, что затраты на переключение просто вычитаются из общей производительности системы.

**6.2.5.5** Если время решения задачи распределено по экспоненциальному закону с параметром $\lambda_{obsl}=1/\bar{t}_{obsl}$, то "оптимальное" значение кванта времени q можно приблизительно рассчитать из условия того, что примерно половина процессов (если они при этом не инициируют запросы ввода-вывода) обслужатся (каждый из них) за квант q, т.е. из условия

$$P\{t_{obsl} < q\} = F(q) - F(0) = 1 - e^{-\lambda_{obsl} \cdot q} = 0{,}5, \qquad (6.1)$$

откуда

$$q = \frac{\ln 2}{\lambda_{obsl}} = \frac{0{,}693}{\lambda_{obsl}} = 0{,}693 \cdot \bar{t}_{obsl}. \qquad (6.2)$$

Очевидно, что формулу (6.2) [соотношение (6.1)] можно использовать для поиска оптимального (вернее, рационального) кванта времени q только в дисциплинах планирования RR и SRT, и то весьма ориентировочно.

**6.2.5.6** При выборе рационального значения кванта времени q (из нескольких возможных) можно руководствоваться также следующими соображениями. Казалось бы, что в качестве рационального кванта времени следует выбрать то его значение, которому соответствует минимальное среднее время реакции системы на запросы ($\bar{t}_{otveta}$). Это будет справедливо лишь в том случае, если разброс времени ответа ($\sigma_{otv}$) у "конкурирующих" между собой значений кванта q будет примерно одинаков. В противном случае можно поступить следующим образом: среди "конкурирующих" значений кванта q выбрать такое значение, для которого соответствующее значение коэффициента вариации $\mathbf{v} = \sigma_{otv}/\bar{t}_{otveta}$ будет минимальным.

**6.2.5.7** Типичное значение кванта в системах разделения времени составляет десятки миллисекунд.

**6.2.6** Дисциплины планирования в системах разделения времени

**6.2.6.1** В СРВ можно выделить следующие "классические" дисциплины планирования работы ЦП:

- циклическое планирование (RR);
- многоуровневые очереди с обратными связями (ТМ);

– планирование по принципу SRT (“по наименьшему остающемуся времени”).

**6.2.6.2** При циклическом или, иначе, круговом планировании (Round Robin – RR) (рисунок 6.1) постановка процессов в очередь готовности осуществляется по принципу FIFO, однако каждый раз запускаемому процессу предоставляется ограниченное количество времени ЦП, называемое временным квантом q. Если процесс не заканчивается до истечения выделенного ему кванта времени ЦП, то ЦП у него отбирается и предоставляется следующему ожидающему (готовому) процессу. Процесс, у которого перехватили ЦП, переходит в конец списка готовых к выполнению процессов. Если процесс заканчивается раньше истечения выделенного ему кванта q, то он завершается, а ЦП занимает следующий процесс из очереди готовых процессов. Таким образом, циклическое планирование (RR) – это, по сути дела, вариант FIFO с переключением ЦП. Такая модель выделения ЦП процессам называется иногда моделью типа "вертушка" (круговоротом) [10].

Дисциплина циклического обслуживания эффективна для работы с

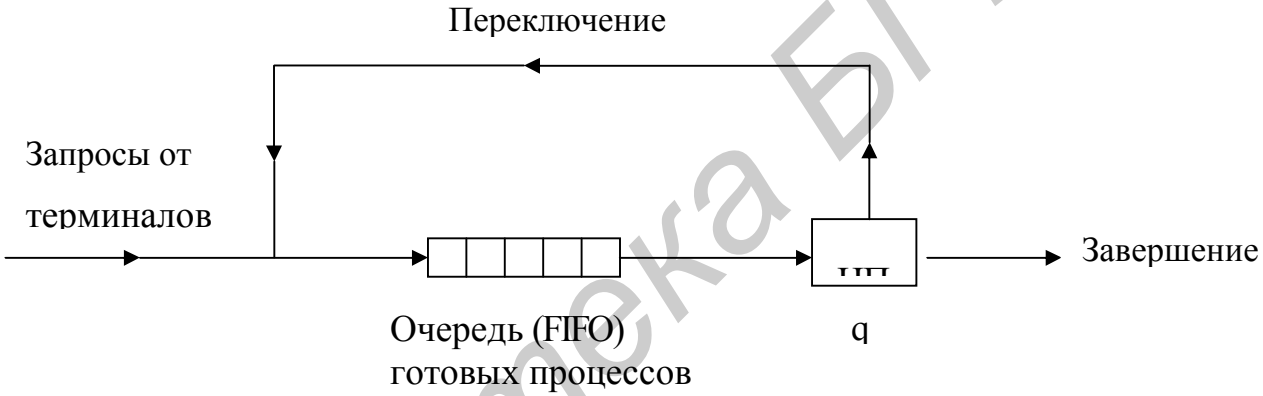



Рисунок 6.1 – Планирование по циклическому принципу (RR)

разделением времени, когда система должна гарантировать приемлемые времена ответа на запросы для всех интерактивных пользователей.

**6.2.6.3** При одновременном обслуживании системой нескольких пользователей, имеющих различные по сложности программы, короткие задания удобно отличать от длинных с целью устранения частых перезапусков последних. Здесь в качестве подходящего механизма как раз и выступают очереди с обратной связью [5, 8, 11].

Многоуровневые очереди с обратной связью состоят из N подочередей. Существует правило, по которому можно определить конкретную подочередь, соответствующую данной программе (процессу). Кроме того, между программами, относящимися к различным подочередям, возникают особые отношения. Принципиальная структура циклической очереди с обратной связью представлена на рисунке 6.2. Процесс начинает выполняться в подочереди 1. Он становится в ней

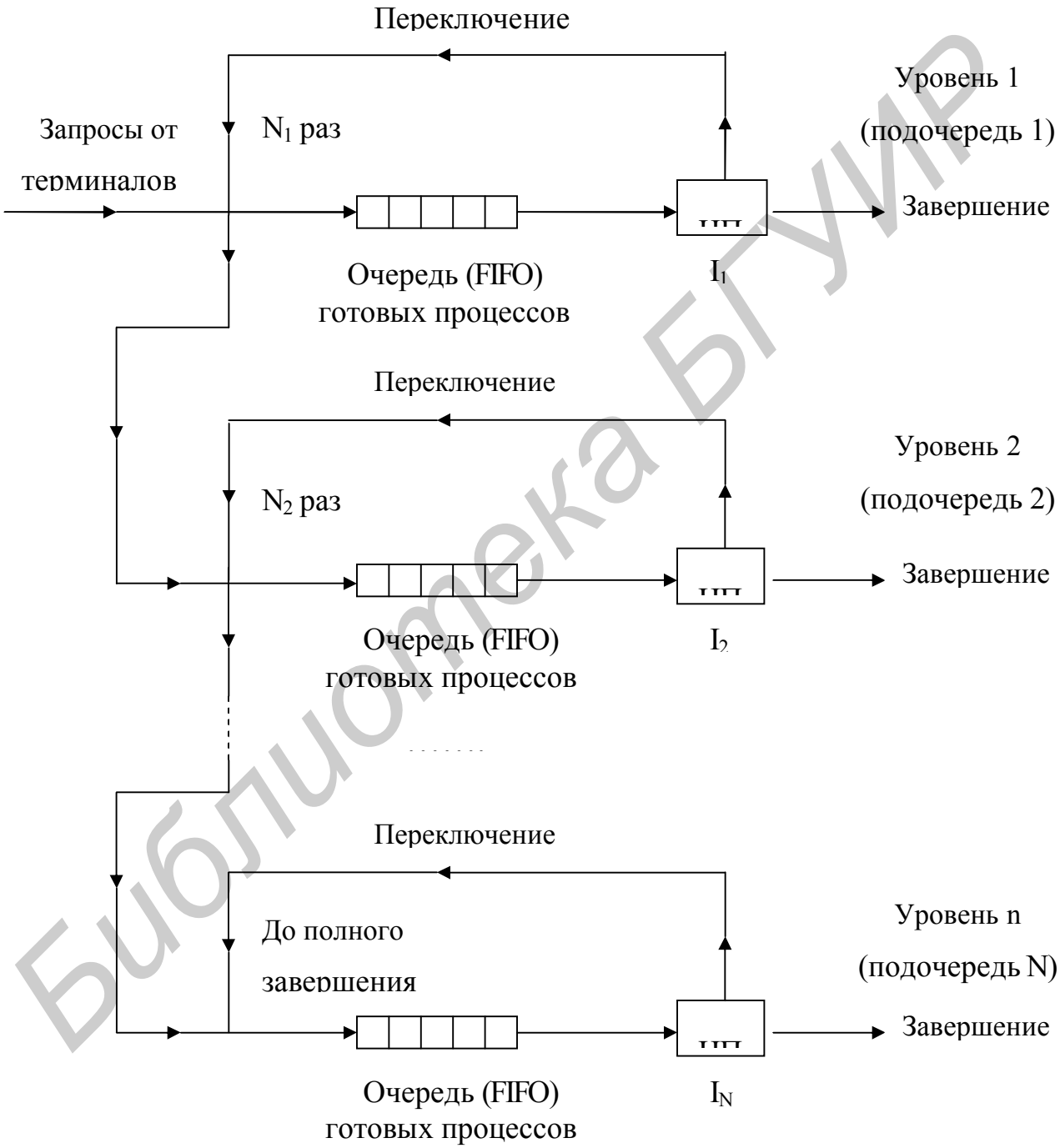



Рисунок 6.2 – Многоуровневые циклические очереди с обратными связями

первым и получает порцию $I_1$ процессорного времени (т.е. времени ЦП). Затем этот процесс перемещается в конец той же подочереди, и со временем снова становится первым и получает очередную порцию $I_1$ времени ЦП. Это повторяется $N_1$ раз, после чего (если, разумеется, процесс не завершился) процесс попадает в подочередь 2. Далее, находясь в подочереди 2, он получает $N_2$ раз порций $I_2$ и перемещается в подочередь 3 и т.д. Попав в N-ю очередь, т.е. в очередь самого нижнего уровня, процесс остаётся в ней до полного завершения. Находясь в подочереди i (i=2,...,N), процесс может получить управление (т.е. занять ЦП) только в том случае, если ни в одной из подочередей с меньшими номерами нет готовых к выполнению программ. В простейшем случае $I_1=I_2=...=I_N=q$; $N_1=N_2=...=N_{N-1}=1$, т.е. процесс в любой очереди, кроме последней, может находиться не более одного раза. Обычно же $N_i>N_j$ и $I_i>I_j$ при i>j. Описанную модель выделения ЦП процессам называют ещё многоуровневой "вертушкой" [10].

Многоуровневые очереди с обратными связями – это идеальный механизм, позволяющий разделять процессы на категории в соответствии с их потребностями во времени ЦП. В системе с разделением времени каждый раз, когда процесс выходит из сети очередей, он может быть помечен признаком очереди самого низкого уровня, в которой он побывал. Когда этот процесс впоследствии вновь войдёт в сеть очередей, он будет направлен прямо в ту очередь, в которой он последний раз завершал свою работу. Здесь планировщик действует на основе следующего эвристического правила: поведение процесса в недавнем прошлом может служить хорошим ориентиром для определения его поведения в ближайшем будущем. Поэтому процесс, лимитируемый ЦП, при своём возвращении в сеть очередей не будет помещаться в очереди более высоких уровней, где он только мешал бы обслуживать короткие процессы высокого приоритета или процессы, лимитируемые вводом-выводом. Если процессы всегда помещать в сеть очередей на самый низкий уровень, который они занимали прошлый раз, то система не сможет реагировать на изменения характера процесса, например на то, что процесс, бывший по преимуществу вычислительным, становится преимущественно "обменным". Эту проблему можно решить, если в метке, которой сопровождается процесс, указывать длительность его выполнения при последнем пребывании в сети очередей. Сеть многоуровневых очередей с обратными связями является примером адаптивного механизма планирования, реагирующего на изменение поведения контролируемой им системы.

**6.2.6.4** Принцип SRT – это аналог принципа SJF [принцип SJF (Shortest Job First, т.е. "кратчайшее задание – первым") является дисциплиной планирования без переключения, согласно которой следующим для выполнения выбирается ожидающее задание с минимальным оценочным рабочим временем, остающимся до завершения], но с переключением, применимый в системах с разделением времени. По принципу SRT всегда выполняется процесс, имеющий минимальное оценочное время до завершения, причём с учётом новых поступающих процессов. По принципу SJF задание, которое запущено в работу, выполняется до завершения. По принципу же SRT выполняющийся процесс может быть прерван при поступлении нового процесса, имеющего более короткое оценочное время работы (рисунок 6.3).

Дисциплина SRT характеризуется более высокими накладными расходами, чем SJF. Механизм SRT должен следить за текущим временем обслуживания выполняющегося задания и обрабатывать возникающие прерывания. Поступающие в систему небольшие процессы будут выполняться почти немедленно. Однако более длительные задания будут иметь даже большее среднее время ожидания и больший разброс времени ожидания (ответа), чем в случае SJF. Реализация принципа SRT требует, чтобы регистрировались истёкшие времена обслуживания, что и приводит к увеличению накладных расходов. Теоретически принцип SRT обеспечивает минимальные времена ожидания, однако из-за издержек на переключения может оказаться так, что в определённых ситуациях в действительности лучшие показатели будет иметь принцип SJF.

Предположим, что поступает задание, оценочное время которого лишь немногим меньше времени, необходимого для завершения текущего задания, причём в основном процессорного времени. В этом случае механизм, реализующий принцип SRT в «чистом» виде, пошёл бы на переключение (прерывание) обслуживаемого процесса. При этом если затраты на переключение превышают разность времён обслуживания обоих заданий, то прерывание текущего задания фактически приведёт к снижению производительности системы. Поэтому для решения данной проблемы можно предусмотреть в системе предотвращение прерывания при возникновении таких ситуаций.

**6.2.6.5** Рассмотренные выше дисциплины планирования весьма условны, поскольку современные ОС с разделением времени используют более изощренные алгоритмы планирования (например, с использованием динамических относительных/абсолютных приоритетов и квантов переменного размера), но все эти алгоритмы являются все же разновидностями дисциплин планирования RR и TM. Что же касается дисциплины планирования SRT, то она перешла в разряд “исторических ценностей”, поскольку требовала знания точного времени обслуживания процессов, что абсолютно неприемлемо для современных ОС, реализующих режим разделения времени (в современных СРВ не используется никакой предварительной информации о задачах). Между тем принцип SRT интересен применением в нем, наряду с квантованием, дисциплины обслуживания с абсолютными приоритетами.

**6.2.7** Исходные данные для модели системы разделения времени

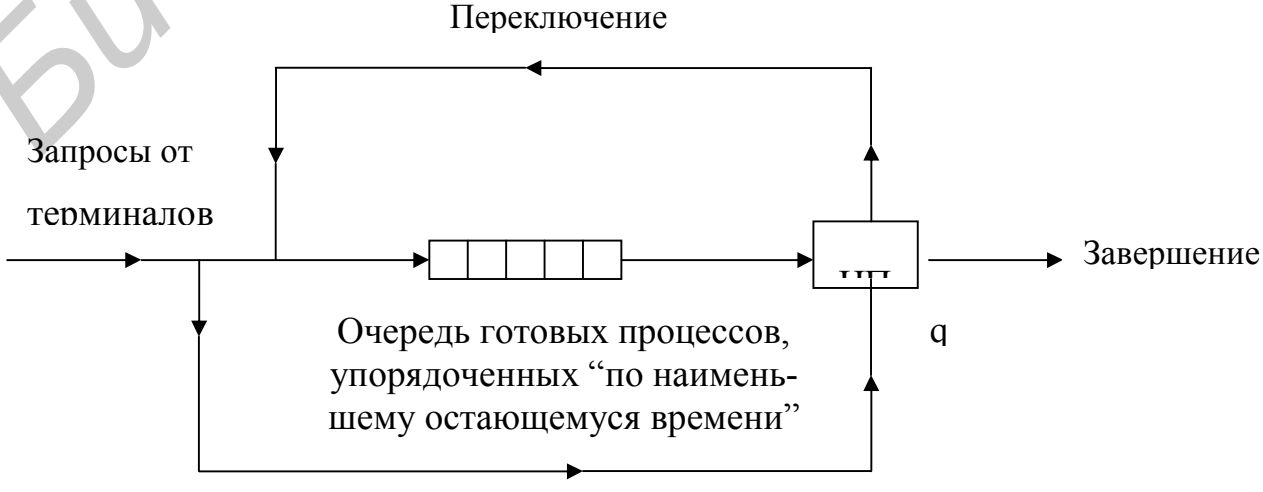



Рисунок 6.3 – Планирование по принципу SRT

**6.2.7.1** Модель системы с разделением времени (независимо от принятой дисциплины планирования) в упрощённом виде может быть построена в виде замкнутой сети (рисунок 6.4), где пользователи в интерактивном режиме (их число постоянно для каждого варианта и равно n) дают системе запросы, которые обслуживает ЦП.

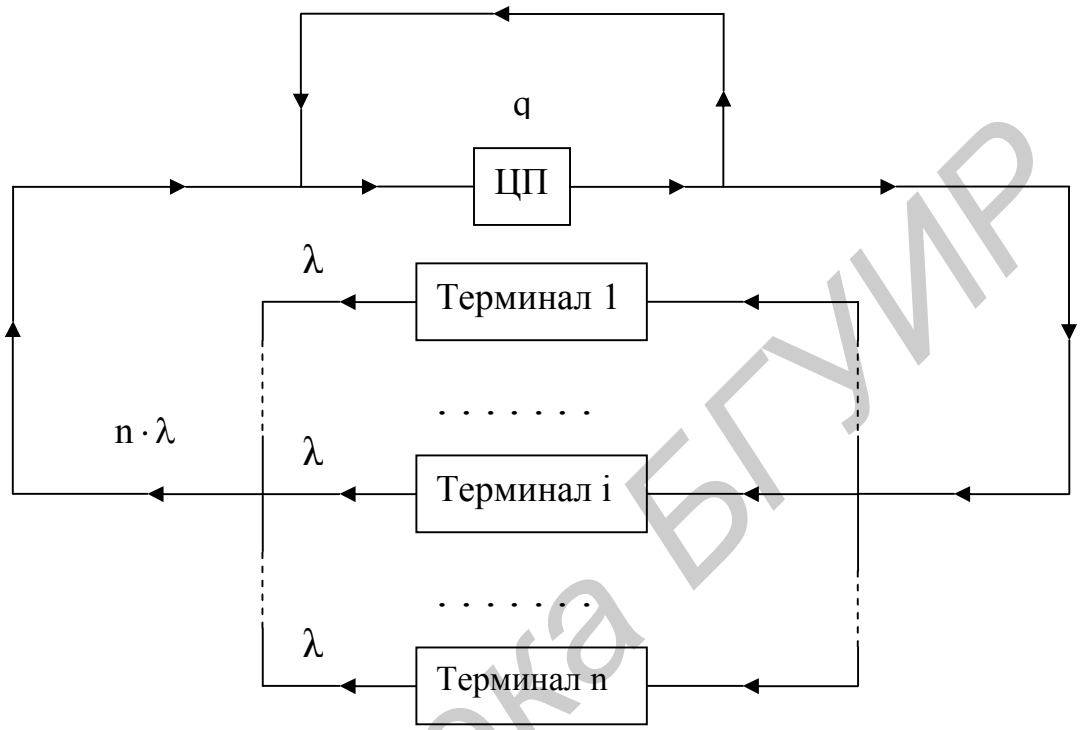



Рисунок 6.4 – Модель системы с разделением времени

Каждому пользователю соответствует процесс. Число терминалов n вы узнаете из результатов моделирования с помощью имитационной модели (оно зависит от номера варианта). Времена формирования запросов ($\bar{t}_{term}$) и их обслуживания ($\bar{t}_{obsl}$) распределены по экспоненциальному закону. Начальные присваиваемые и вычисляемые значения параметров модели СРВ с разными дисциплинами планирования следующие:

$\bar{t}_{term} = 5$ с; $\lambda = 1/\bar{t}_{term}$ с$^{-1}$; **q**=10 мс; $\bar{t}_{obsl} = 100$ мс;

$t_{perek} = 1$ мс; $\mathbf{N_i}=\mathbf{i}$; $\mathbf{I_i}=\mathbf{i}*\mathbf{q}$.

**6.2.7.2** Пороговое значение времени в планировании по принципу SRT принято равным $t_{perek}$. Варьируемые параметры для каждой конкретной модели будут указаны в порядке выполнения работы. Исходное время моделирования –2 мин (120000000 мкс) (в зависимости от типа ПЭВМ его можно увеличивать в разумных пределах для увеличения точности результатов моделирования). В GPSS-моделях все временные параметры заданы в микросекундах (мкс).

## 6.3 Порядок выполнения работы

**6.3.1** Часть 1

**6.3.1.1** Получить у преподавателя номер варианта исходных данных для модели, представленной на рисунке 6.4.

**6.3.1.2** Скопировать в свой рабочий каталог D:\OS из каталога D:\OS\LABOS\LABOS6\CHAST_1 файлы modos6a.gps, modos6a1.gps, lab6a.bat и startup.gps. Файл modos6a.gps содержит исходный текст имитационной модели на языке GPSS/PC, реализующей систему разделения времени с циклическим планированием (RR). Предварительно в рабочем каталоге должны быть размещены следующие обязательные для работы файлы: gpsspc.exe, gpssrept.exe, settings.gps и position.gps (из каталога D:\OS\GPSS).

**6.3.1.3** Изменить содержание поля операнда оператора (из файла modos6a.gps)

VARIANT VARIABLE номер_варианта

в соответствии со своим вариантом.

**6.3.1.4** Получить результаты моделирования с помощью имитационной модели, при этом автоматизировать работу с моделью, запустив соответствующий batch-файл lab6a.bat с учётом содержимого файла startup.gps.

**6.3.1.5** Добиться оптимального для условий Вашего варианта значения временного кванта q (см. пункт 6.2.5), фиксируя при этом все промежуточные результаты. Изменение временного кванта q производится посредством изменения поля операнда следующего оператора (из файла modos6a.gps)

Q_KVANT VARIABLE временной_квант_q_в_микросекундах.

Обосновать выбор размера оптимального кванта.

**6.3.1.6** Построить зависимости среднего времени ответа (реакции системы) на запрос и накладных расходов системы разделения времени (доли коэффициента загрузки ЦП, вызванной затратами процессорного времени на переключение, в общем коэффициенте загрузки ЦП) в функции временного кванта q.

**6.3.1.7** Запустить модель с дисциплиной FIFO (файл modos6a1.gps), предварительно задав полученный ранее номер своего варианта (аналогично подпункту 6.3.1.3) и внеся соответствующие изменения в batch-файл и файл startup.gps.

**6.3.1.8** Сравнить выходные результаты модели с дисциплиной планирования FIFO (т.е. без переключения ЦП) с выходными результатами модели с дисциплиной обслуживания RR

**6.3.1.9** Оформить отчет и сформулировать выводы по работе (в виде текстового файла).

**6.3.2** Часть 2

**6.3.2.1** Получить у преподавателя номер варианта исходных данных для модели, представленной на рисунке 6.4 (он совпадает с номером варианта из первой части работы).

**6.3.2.2** Скопировать в свой рабочий каталог D:\OS из каталога D:\OS\LABOS\LABOS6\CHAST_1 файл modos6a2.gps, а из каталога D:\OS\LABOS\LABOS6\CHAST_2 – файлы modos6b.gps, modos6c.gps, lab6b.bat, lab6c.bat и startup.gps. Файл modos6b.gps содержит исходный текст имитационной модели на языке GPSS/PC, реализующей систему разделения времени с планированием на основе многоуровневых очередей с обратными связями. Предварительно в рабочем каталоге должны быть размещены следующие обязательные для работы файлы: gpsspc.exe, gpssrept.exe, settings.gps и position.gps (из каталога D:\OS\GPSS).

**6.3.2.3** Изменить содержание поля операнда оператора (из файла modos6b.gps)

VARIANT VARIABLE номер_варианта

в соответствии со своим вариантом.

**6.3.2.4** Получить результаты моделирования с помощью имитационной модели, при этом автоматизировать работу с моделью, запустив соответствующий batch-файл lab6b.bat с учётом содержимого файла startup.gps.

**6.3.2.5** Проанализировать полученные результаты (время ответа, средние длины подочередей, разброс времени ответа и т.п.) и сделать соответствующий им вывод.

**6.3.2.6** Добиться оптимального для условий Вашего варианта значения временного кванта q при неизменном числе подочередей (N=3), фиксируя при этом все промежуточные результаты. Изменение временного кванта q производится посредством изменения поля операнда (из файла modos6b.gps)

Q_KVANT VARIABLE временой_квант_q_в_микросекундах,

при этом не пропустить задания оптимального значения q для модели modos6a.gps, реализующей циклическое планирование (RR) (см. подпункт 6.3.1.5). Обосновать выбор размера оптимального кванта q.

**6.3.2.7** Сравнить между собой выходные параметры моделей modos6a.gps и modos6b.gps при соответствующих им оптимальных временных квантах q. Сделать выводы.

**6.3.2.8** Установить в модели modos6b.gps начальное значение кванта q (см. пункт 6.2.8), изменив поле операнда оператора

Q_KVANT VARIABLE временной_квант_q_в_микросекундах .

**6.3.2.9** Добиться оптимального для условий Вашего варианта числа подочередей N при неизменном начальном значении кванта q (q=10мс), фиксируя при этом все промежуточные результаты. Изменение числа подочередей N производится посредством изменения поля операнда следующего оператора (из файла modos6b.gps):

N_OCHER VARIABLE число_подочередей .

Обосновать выбор оптимального числа подочередей ($N_{opt} \geq 2$). Сравнить между собой результаты моделирования, полученные с помощью модели modos6b.gps в подпунктах 6.3.2.6 и 6.3.2.9, и сделать выводы.

**6.3.2.10** Получить у преподавателя номер варианта исходных данных для модели, представленной на рисунках 6.3, 6.4 (он должен совпадать с номером варианта, полученным при выполнении первой части данной лабораторной работы).

**6.3.2.11** Скопировать в свой рабочий каталог файл modos6c.gps, содержащий исходный текст имитационной модели на языке GPSS/PC, реализующей систему разделения времени с планированием по принципу SRT.

**6.3.2.12** Изменить содержание поля операнда оператора (из файла modos6c.gps)

VARIANT VARIABLE номер_варианта

в соответствии со своим вариантом.

**6.3.2.13** Получить результаты моделирования с помощью имитационной модели, изменив предварительно в файле startup.gps имя modos6b на modos6c и запустив соответствующий batch-файл lab6c.bat, а.

**6.3.2.14** Проанализировать полученные результаты (время ответа, средние длины подочередей, разброс времени ответа и т.п.) и сделать соответствующий им вывод.

**6.3.2.15** Добиться оптимального для условий Вашего варианта значения временного кванта q, фиксируя при этом все промежуточные результаты. Изменение временного кванта q производится посредством изменения поля операнда (из файла modos6c.gps)

Q_KVANT VARIABLE временной_квант_q_в_микросекундах ,

при этом не пропустить задания оптимального значения q для модели modos6a.gps, реализующей циклическое планирование (RR) (см. подпункты 6.2.7.2 и 6.3.1.5). Обосновать выбор размера оптимального кванта.

**6.3.2.16** Сравнить между собой выходные параметры моделей modos6a.gps и modos6c.gps при соответствующих им оптимальных временных квантах q. Сделать выводы.

**6.3.2.17** Запустить модель с дисциплиной SJF (файл modos6a2.gps), предварительно задав полученный ранее номер своего варианта и внеся соответствующие изменения в соответствующий batch-файл и в файл startup.gps (взять любой batch-файл и все имена заменить на modos6a2; то же самое сделать и в файле startup.gps).

**6.3.2.18** Сравнить выходные результаты модели с дисциплиной обслуживания SJF (т.е. без переключения ЦП) с выходными результатами модели с дисциплиной обслуживания SRT (причем с оптимальным значением временного кванта q).

**6.3.2.19** Оформить отчет (дополнить часть 1) и сделать общие выводы по работе (в виде текстового файла).

## 6.4 Контрольные вопросы

**6.4.1** Поясните суть режима разделения времени.

**6.4.2** В чём различие мультипрограммирования и режима разделения времени?

**6.4.3** Исключает ли режим разделения времени мультипрограммный режим?

**6.4.4** Какие бывают приоритеты процессов в ВС?

**6.4.5** В каких случаях происходит смена активного процесса в СРВ?

**6.4.6** Что понимается под накладными расходами в системах с квантованием?

**6.4.7** Из каких соображений выбирается размер временного кванта?

**6.4.8** Поясните суть циклического планирования RR.

**6.4.9** Поясните суть многоуровневых очередей с обратными связями (многоуровневых "вертушек").

**6.4.10** В чём проявляется адаптация многоуровневых очередей с обратными связями к изменению потребности процессов во времени ЦП?

**6.4.11** Как обычно изменяются $N_i$ и $I_i$ при возрастании номера i подочереди в СРВ с многоуровневыми очередями с обратными связями?

**6.4.12** Поясните суть планирования по принципу SRT.

**6.4.13** В чём отличие принципа SRT от принципа SJF?

**6.4.14** В чём недостаток "чистого" механизма SRT и как его устранить?

**6.4.15** В какой очереди (ожидающих или готовых) скапливается большее число процессов: в интерактивных системах разделения времени или в системах пакетной обработки, решающих “счетные” задачи?

**6.4.16** Известно, что программа А выполняется в монопольном режиме за 10 мин, а программа В – за 20 мин, т.е. при последовательном выполнении они требуют 30 мин. Если Т – время выполнения обеих этих задач в режиме мультипрограммирования, то какое из неравенств, приведенных ниже, справедливо:

а) $T<10$;

б) $10<T<20$;

в) $20<T<30$;

г) $T>30$ ?

**6.4.17** Могут ли быть применены сразу все перечисленные (в одном пункте перечисления) характеристики к одному алгоритму планирования процессов:

а) вытесняющий, с абсолютными динамическими приоритетами;

б) невытесняющий, с абсолютными фиксированными приоритетами;

в) невытесняющий, с относительными динамическими приоритетами;

г) вытесняющий, с абсолютными фиксированными приоритетами, основанный на квантовании с динамически изменяющимся размером кванта;

д) невытесняющий, основанный на квантовании с фиксированной длиной кванта ?

**6.4.18** Обозначив среднее время контекстного переключения при переходе с процесса на процесс через s, а среднее количество времени, которое использует процесс, лимитируемый вводом-выводом, прежде чем сформирует запрос ввода-вывода, через t ($t \gg s$), обсудите, к каким последствиям приведет выбор каждого из следующих значений кванта времени:

а) $q = \infty$;
б) q несколько больше нуля;
в) $q = s$;
г) $s < q < t$;
д) $q = t$;
е) $q > t$.

# 7 КОМПЛЕКС МОДЕЛЕЙ ОБРАБОТКИ ВЗАИМНЫХ БЛОКИРОВОК

## 7.1 Цель работы

**7.1.1** Получить сведения о причинах возникновения тупиковых ситуаций в ВС и подходах к разрешению проблемы тупиков.

**7.1.2** Изучить с помощью имитационных моделей влияние различных стратегий разрешения проблемы тупиков на эффективность работы ВС.

## 7.2 Тупиковые ситуации и подходы к их разрешению

**7.2.1** Понятие тупика

**7.2.1.1** При параллельном исполнении процессов могут возникать такие ситуации, при которых два и более процесса всё время находятся в состоянии блокировки. О таких процессах говорят, что они находятся в состоянии *взаимной блокировки*, *тупика*, *дедлока* (deadlock) или *клинча* (clinch) ("*смертельного объятия*" – deadly embrace). С проблемой тупиков тесно связана проблема *бесконечного откладывания*, когда процесс, даже не находящийся в состоянии тупика, ожидает события, которое может никогда не произойти из-за "необъективных" принципов, заложенных в системе планирования ресурсов.

**7.2.1.2** Хотя тупик и может быть результатом ошибок программирования, но чаще всего он возникает не из-за них. Поскольку ОС выполняет по преимуществу функции администратора ресурсов, то и проблема тупиков чаще всего связана с распределением ресурсов между несколькими процессами, причём возникновение тупиков в основном связано не с общим разделяемым ресурсом (используемым одновременно несколькими процессами), а с выделяемыми, или закрепляемыми, ресурсами (т.е. с ресурсами, которые в каждый момент времени отводятся только одному процессу и которые поэтому иногда называются ресурсами последовательного использования). Одним из самых простых примеров тупика при распределении ресурсов является случай, когда каждый из двух процессов ждет ресурс, занятый другим процессом; из-за этого ожидания ни один из процессов не может продолжить исполнение и освободить ресурс, необходимый другому процессу (рисунок 7.1) [5,8–11,13].

Стоимость дедлока велика. Ни процесс X, ни процесс Y не могут закончить выполнение. Более того, все ресурсы, которые получили оба процесса (по крайней мере ресурсы P1 и P2), становятся недоступными, что снижает возможность системы обслуживать другие процессы.

**7.2.2** Условия возникновения тупиков

**7.2.2.1** Для возникновения тупика необходимо, чтобы *одновременно* выполнялись четыре условия (их сформулировали Коффман, Элфик и Шошани):

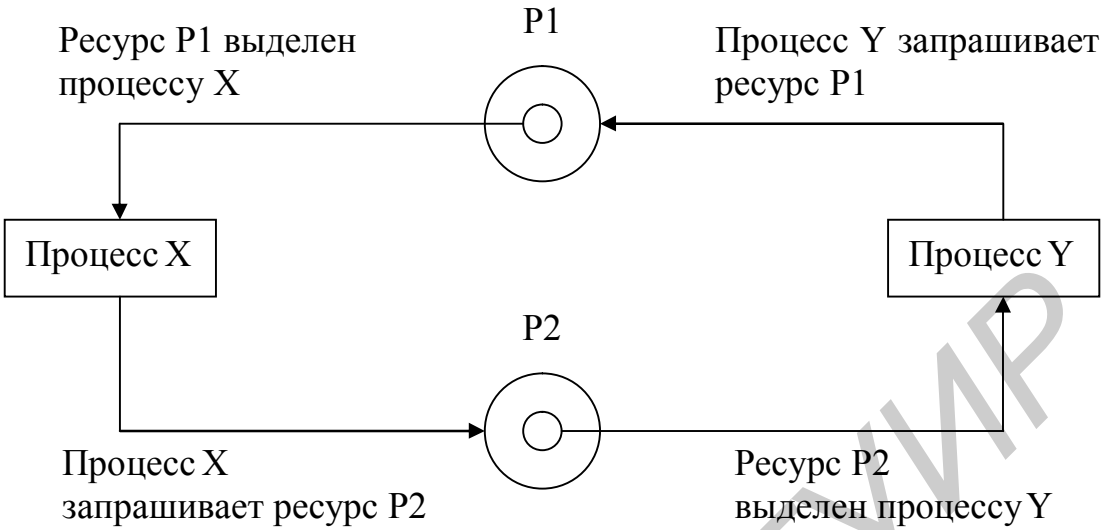



Рисунок 7.1– Простая тупиковая ситуация при распределении ресурсов

а) условие взаимного исключения (процессы осуществляют монопольный доступ к ресурсам);

б) условие ожидания ресурсов (процесс, запросивший ресурс, будет ждать, пока запрос не будет удовлетворен, продолжая при этом удерживать все остальные ресурсы, которые он уже получил);

в) условие неперераспределяемости ресурсов (никакие ресурсы нельзя отобрать у процесса, если они ему уже выделены);

г) условие кругового ожидания (существует замкнутая цепь процессов, каждый из которых ждет ресурс, удерживаемый его предшественником в этой цепи).

**7.2.3** Подходы к разрешению проблемы тупиков

**7.2.3.1** Невозможность процессов завершить начатую работу из-за возникновения взаимных блокировок снижает производительность ВС. Поэтому проблеме тупиков уделяется большое внимание. Для разрешения проблемы тупиковых ситуаций можно воспользоваться одной из следующих стратегий:

– стратегией *предотвращения* (*предупреждения*) тупиков;

– стратегией *обхода* тупиков;

– стратегией *распознавания* (*обнаружения*) тупиков и последующего *восстановления* .

**7.2.3.2** Стратегия предотвращения дедлоков исходит из того, что они настолько дорогостоящи, что лучше потратить дополнительные ресурсы системы, чтобы исключить вероятность возникновения тупика при любых обстоятельствах. Стратегия обхода дедлоков гарантирует, что тупик, хотя он в принципе и возможен, не возникает для конкретного набора процессов и запросов, выполняющихся в данный момент. Стратегия распознавания дедлоков и последующего восстановления

базируется на том, что дедлок возникает достаточно редко и что дороже предотвращать или обходить возможность его появления, чем распознать его и провести восстановление. Предотвращение тупиков можно рассматривать как запрет существования опасных состояний ВС, обход – как запрет входа в опасное состояние, а восстановление – как запрет постоянного пребывания в опасном состоянии. Ниже рассматриваются дисциплины для реализации каждой из этих стратегий.

**7.2.3.3** Предотвращение тупиков имеет целью обеспечение условий, исключающих возможность возникновения тупиковых ситуаций. Такой подход является вполне корректным решением в том, что касается самого тупика, однако он часто приводит к нерациональному использованию ресурсов. Тем не менее различные методы предотвращения тупиков широко применяются в практике разработчиков. Хавендер показал, что возникновение тупика невозможно, если нарушено хотя бы одно из четырех необходимых условий его возникновения (см. пункт 7.2.2). Для предотвращения тупиков он предложил следующие три дисциплины:

– *предварительное (статическое) распределение ресурсов* (иначе, *метод глобального распределения ресурсов*) (нарушение условия ожидания дополнительных ресурсов) предполагает, что каждый процесс должен потребовать все требуемые ему ресурсы заранее и он не сможет начать исполнение до тех пор, пока они ему не будут выделены. В таком случае общее число ресурсов, необходимое каждому из параллельных процессов, не может превышать возможностей системы. Процесс может освободить ресурс, если тот становится ему не нужен. Когда процесс находится в состоянии ожидания всех необходимых ему ресурсов, то он не должен удерживать какие-либо ресурсы; благодаря этому предотвращается возникновение условия ожидания дополнительных ресурсов и тупиковые ситуации просто невозможны. Этот метод приводит к непродуктивному замораживанию ресурсов и на практике часто исключает всякий параллелизм в выполнении. Ещё одним недостатком является тот факт, что трудно сформулировать запросы на ресурсы в тех случаях, когда требуемые процессу ресурсы становятся известны только после начала исполнения;

– *общее исключение* (нарушение условия неперераспределяемости ресурсов) позволяет ОС отнимать у процесса ресурсы; это выполнимо, если можно запомнить состояние процесса для его последующего восстановления. Эта дисциплина предполагает, что если процесс, удерживающий определенные ресурсы, получает отказ в удовлетворении запроса на дополнительные ресурсы, то данный процесс должен освободить *все* свои первоначальные ресурсы и при необходимости запросить их снова вместе с дополнительными. Но этот способ предотвращения дедлоков также не свободен от недостатков: если процесс в течение некоторого времени использует определенные ресурсы, а затем освобождает их, то он может потерять всю работу, проделанную до данного момента. Если такая ситуация встречается редко, то можно считать, что в нашем распоряжении имеется относительно недорогой способ предотвращения тупиков; если же такая ситуация встречается часто, то подобный способ обходится дорого, причем приводит к

печальным результатам, особенно когда не удается вовремя завершить высокоприоритетные или срочные процессы. Еще одним из серьезных недостатков такой дисциплины является возможность бесконечного откладывания процессов: выполнение процесса, который многократно запрашивает и освобождает одни и те же ресурсы, может откладываться на неопределенно долгий срок;

– *иерархическое выделение ресурсов (распределение по стандартной последовательности*, *или метод упорядоченных классов)* (нарушение условия кругового ожидания) предполагает введение некоторой иерархии ресурсов, и процесс, затребовавший ресурс на одном уровне, может затем потребовать ресурсы только на более высоком уровне. Далее он (процесс) может освободить ресурсы на данном уровне только после освобождения всех ресурсов на всех более высоких уровнях. Только после того как процесс получил, а потом освободил ресурсы данного уровня, он может снова запросить ресурсы на том же самом уровне. Предварительное выделение ресурсов можно считать специальным случаем иерархического выделения, имеющего единственный уровень. Иерархическое выделение несколько дороже, но оно может снизить потери, связанные с полным предварительным выделением. Однако этот метод не дает никакого выигрыша, если порядок, в котором процессам необходимы ресурсы, отличается от порядка уровней в иерархии.

Заметим, что Хавендер предложил три стратегии, а не четыре (число необходимых условий возникновения тупика). Первое необходимое условие – условие взаимоисключения, согласно которому процессы получают право на монопольное управление выделяемыми им ресурсами, мы не вправе нарушать, поскольку нам нужно предусмотреть возможность работы с закрепленными ресурсами (хотя разрешение неограниченного разделения ресурсов очень удобно для таких ресурсов, как повторно входимые программы, например редакторы).

**7.2.3.4** Цель средств обхода тупиков заключается в том, чтобы можно было предусматривать менее жесткие ограничения, чем в случае предотвращения тупиков, и тем самым обеспечить лучшее использование ресурсов. Методы обхода тупиков учитывают возможность их возникновения, однако в случае увеличения вероятности конкретной тупиковой ситуации здесь принимаются меры по аккуратному обходу тупика. Наиболее известным алгоритмом обхода тупиковых ситуаций является *алгоритм банкира*, предложенный Дейкстрой и осуществляющий контролируемое выделение ресурса. Каждый процесс предоставляет заранее *анонс*, т.е. верхнюю границу своих запросов на каждый ресурс, а алгоритм банкира позволяет избежать тупика, пересчитывая риск при каждом выделении ресурсов. Операционная система примет запрос процесса в том случае, если анонс этого процесса не превышает соответствующего числа располагаемых ВС ресурсов каждого вида. Если ОС со своей стороны в состоянии удовлетворить максимальную потребность процесса в ресурсе, то процесс со своей стороны гарантирует, что эти ресурсы будут использованы и возвращены ОС в течение конечного периода времени. Алгоритм банкира базируется на понятии надежного (безопасного) состояния: состояние системы является *надежным*, если, исходя из него, можно заставить систему функционировать без тупиков при самых пессимистических предположениях, т.е.

когда каждый процесс действительно запрашивает максимальное число предусмотренных ресурсов; в противном случае текущее состояние системы называется *ненадежным* (опасным). Алгоритм банкира выполняется при каждом запросе ресурса, и ресурс выделяется только в том случае, если это выделение не выводит систему из надежного состояния. Подчеркнем, что в надежном состоянии остаток потребности в ресурсах всегда должен быть хотя бы для одного процесса (из n процессов) не больше числа свободных единиц ресурсов. Приведем следующий пример. Пусть в системе имеется 12 единиц ресурса одного типа, которые распределяются между тремя процессами. Исходное состояние системы – состояние I – представлено в таблице 7.1.

Таблица 7.1 – Состояние I

| Процессы | Текущее количество выделенного ресурса | Резерв | Максимальная потребность (анонс) |
|---|---|---|---|
| Процесс 1 | 1 | | 4 |
| Процесс 2 | 4 | | 6 |
| Процесс 3 | 5 | | 8 |
| Итого | 10 | 2 | – |

Состояние I (см. таблицу 7.1) является надежным, поскольку оно дает возможность всем трем процессам завершиться. Если даже всем процессам одновременно понадобится максимальное число единиц ресурса, то есть выход: отдать резерв процессу 2. После завершения процесс 2 освободит все 6 единиц ресурса, так что система сможет выделить их процессу 1 и процессу 3 (одновременно по три единицы каждому или по 3 последовательно первому, затем – третьему или наоборот). Таким образом, основной критерий надежного состояния – это существование последовательности действий, позволяющей процессам завершиться. Если известно, что данное состояние надежно, то это вовсе не означает, что все последующие состояния также будут надежными. Предположим теперь, что процесс 3 запрашивает дополнительный ресурс. Если бы система удовлетворила этот запрос, то она перешла бы в новое состояние II (таблица 7.2).

Таблица 7.2 – Состояние II

| Процессы | Текущее количество выделенного ресурса | Резерв | Максимальная потребность (анонс) |
|---|---|---|---|
| Процесс 1 | 1 | | 4 |
| Процесс 2 | 4 | | 6 |
| Процесс 3 | 6 | | 8 |
| Итого | 11 | 1 | – |

Состояние II (см. таблицу 7.2) является ненадежным, поскольку система, попадая в него, не может гарантировать успешное завершение всех процессов пользователей. Если всем трем процессам для продолжения работы понадобится более одной единицы ресурса, то возникнет тупик. Важно отметить, что термин

"ненадежное состояние" не предполагает, что в данный момент существует или в другой момент времени обязательно возникнет тупиковая ситуация. Он просто говорит о том, что в случае некоторой неблагоприятной последовательности событий система может зайти в тупик. Таким образом, согласно алгоритму банкира система удовлетворяет только те запросы, при которых ее состояние остается надежным. Запрос процесса, приводящий к переходу системы в ненадежное состояние, откладывается до момента, когда его всё же можно будет выполнить, причем процесс удерживает за собой уже выделенные ему ресурсы. Следовательно, поскольку система всегда поддерживается в надежном состоянии, рано или поздно (т.е. в течение конечного времени) все запросы будут удовлетворены и все процессы смогут завершиться.

Основные недостатки алгоритма банкира следующие:

– требуются анонсы процессов (трудно заранее указать, какие ресурсы и в каком количестве потребуются);

– фиксированное количество распределяемых ресурсов (трудно поддерживать);

– постоянное число работающих пользователей (процессов) (оно непрерывно меняется);

– требуется гарантия конечности использования ресурсов пользователями (для реальных систем требуются гораздо более конкретные гарантии).

**7.2.3.5** Обнаружение тупика – это установление факта, что возникла тупиковая ситуация, и определение процессов и ресурсов, вовлеченных в эту тупиковую ситуацию. Алгоритмы обнаружения тупиков, как правило, применяются в системах, где выполняются первые три необходимых условия возникновения тупика, и определяют, не создался ли режим кругового ожидания освобождения ресурсов. Алгоритм обнаружения (алгоритм распознавания замкнутых цепей) можно выполнять с любой нужной частотой (часто – всякий раз, когда запрос на ресурс отклоняется, или редко – раз в час), но в любом случае его применение сопряжено с дополнительными затратами машинного времени.

После обнаружения тупика должно быть выполнено восстановление нормальной работы системы. Это обязательно вызовет перезапуск одного или нескольких процессов, вовлеченных в дедлок. Процессы, находящиеся в тупике, были заблокированы в некоторой точке, и их надо вернуть в те условия (точки), в которых они могут возобновить свое исполнение. Возврат процессов к "точкам", которые предшествовали запросам, приведшим к тупику, можно выполнить с помощью операции "*контрольная точка*". Дело в том, что процесс может сделать снимок своего исполнения, т.е. запомнить информацию о своем состоянии в некоторой контрольной точке. В этом случае при повторном запуске этого процесса не потребуется повторять все вычисления, предшествовавшие дедлоку, а он запустится с последней контрольной точки. Следует заметить, что контрольные точки обычно вводятся не для того, чтобы помочь восстановлению после тупика, а для того, чтобы продолжить исполнение после возникновения ошибки (особенно при выполнении длинных работ в системах реального времени). Перечислим способы восстановления:

– принудительно завершать все процессы и запускать ОС заново (самый простой и самый дорогой в смысле потерь способ);

– принудительно завершать все процессы, находящиеся в дедлоке (в этом случае пользователи могут запустить их когда-нибудь снова);

– принудительно завершать процессы, находящиеся в дедлоке, по одному и после каждого завершения вызывать алгоритм обнаружения тупика (или обращаться для этого к оператору) до тех пор, пока дедлок не исчезнет;

– с помощью контрольных точек (при этом процессы, находящиеся в дедлоке, запускаются со своих контрольных точек в предположении, что тупик больше не возникнет);

– перераспределять ресурсы одного или нескольких процессов, среди которых могут быть даже не вовлеченные в дедлок процессы (ресурсы назначаются одному из оставшихся процессов, находящихся в дедлоке, и он возобновляет исполнение);

– если в момент, когда дедлок выявлен, существует несколько пользовательских процессов, которые не находятся в дедлоке, то этим процессам позволяется доработать до конца при запрещении образования новых процессов и тогда они могут освободить достаточно ресурсов, чтобы "разорвать" дедлок.

Стоимость стратегии распознавания дедлока зависит от того, насколько часто выполняется алгоритм распознавания (обнаружения). Основная "цена" восстановления после дедлока – это потери времени, которые могут быть существенными. Если автоматическое обнаружение тупиков (т.е. с помощью соответствующего алгоритма) и восстановление после них дают эффект при сравнительно редком возникновении тупиков, то при очень редком их появлении можно обнаруживать тупики с помощью оператора системы, а бороться с ними рестартом системы.

**7.2.3.6** Тупики по мере развития ВС становятся всё более критическим фактором, поскольку в них (ВС) возрастает доля динамического распределения ресурсов и количество одновременно выполняемых процессов.

## 7.3 Исходные данные для моделей разрешения проблемы тупиков

**7.3.1** Предварительное распределение ресурсов

**7.3.1.1** Модель предварительного распределения ресурсов реализована в виде GPSS-модели (файл modos71.gps) со следующими исходными данными:

– два типа процессов X и Y поступают в мультипроцессорную систему, содержащую два процессора, и используют при своем выполнении два типа ресурсов – P1 и P2;

– система содержит 7 единиц ресурса P1 и 6 единиц ресурса P2;

– для выполнения процессу X может быть необходимо от 1 до 6 единиц ресурса P1 и от 1 до 3 единиц ресурса P2 (потребность в ресурсе разыгрывается

случайным образом), для процесса Y – от 1 до 4 единиц ресурса P1 и от 1 до 5 единиц ресурса P2;

– процесс развивается только после получения всех затребованных ресурсов, а освобождать ресурсы он может и не одновременно;

– время работы над каждым типом ресурса имитируется случайным образом;

– порядок использования ресурсов у процессов типа X следующий: P1→P2, у процессов типа Y – обратный, при этом первый используемый процессом тип ресурса либо может после работы с ним освобождаться, либо может быть закреплен за процессом до его завершения (оба этих исхода равновероятны).

**7.3.1.2** Номер варианта влияет на интенсивность поступления процессов в систему, на время работы над каждым типом ресурса, а также на время моделирования.

**7.3.2** Общее исключение

**7.3.2.1** Модель общего исключения реализована на языке GPSS/PC (файл modos72.gps) на основе исходных данных из пункта 7.3.1 с учетом следующего замечания: если процесс, удерживающий определенные ресурсы, получает отказ в удовлетворении запроса на дополнительные ресурсы, то этот процесс должен освободить все занимаемые им ресурсы и запросить их снова вместе с дополнительными, при этом вся работа, проделанная до момента отказа в выделении ресурса, считается потерянной.

**7.3.3** Иерархическое выделение ресурсов

**7.3.3.1** Модель иерархического выделения ресурсов реализована на языке GPSS/PC (файл modos73.gps) на основе исходных данных из пункта 7.3.1 с учетом следующего замечания: ресурсы первого типа, т.е. P1, отнесем к классу 1, а ресурсы второго типа, т.е. P2, – к классу 2, следовательно, процессы (и X, и Y) захватывают ресурсы в последовательности P1→P2, а освобождают – в последовательности P2→P1. Если процесс получает отказ в выделении ресурса P2, то он, не освобождая ресурс P1, освобождает процессор и становится в очередь к ресурсу P2.

**7.3.4** Алгоритм банкира

**7.3.4.1** Модель, реализующая алгоритм банкира на языке GPSS/PC (файл modos74.gps), основывается на следующих исходных данных:

– для того чтобы построить комбинацию согласно алгоритму, возьмем конечное число процессов – 6, которые будут использовать два типа ресурсов P1 и P2;

– система содержит 7 единиц ресурса P1 и 6 единиц ресурса P2;

– для выполнения процесс может запросить любое количество ресурсов каждого типа, однако известны максимально возможные потребности в ресурсах каждого типа, автоматически задаваемые в модели через задание своего номера варианта.

**7.3.4.2** В таблице 7.3 представлены исходные данные для построения (по ходу выполнения работы) последовательности надежных состояний системы при удовлетворении запросов процессов на ресурсы по алгоритму банкира.

Таблица 7.3

| Вариант | Анонс процессов (и последовательность запрашиваемых единиц ресурсов) | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Процесс 1 | | Процесс 2 | | Процесс 3 | | Процесс 4 | | Процесс 5 | | Процесс 6 | |
| | P1 | P2 | P1 | P2 | P1 | P2 | P1 | P2 | P1 | P2 | P1 | P2 |
| 1 | 4(4+0+0) | 3(1+1+1) | 6(5+0+1) | 3(0+0+3) | 2(1+0+1) | 2(0+0+2) | 5(4+0+1) | 4(1+1+2) | 4(4+0+0) | 2(0+1+1) | 2(1+0+1) | 2(2+0+0) |
| 2 | 2(2+0+0) | 2(0+2+0) | 4(4+0+0) | 6(5+1+0) | 3(0+3+0) | 3(0+2+1) | 4(0+3+1) | 2(1+0+1) | 5(1+4+0) | 4(3+1+0) | 2(0+0+2) | 4(0+0+4) |
| 3 | 3(1+1+1) | 2(2+0+0) | 6(6+0+0) | 3(3+0+0) | 4(1+1+2) | 4(0+4+0) | 2(2+0+0) | 3(1+2+0) | 2(2+0+0) | 2(0+2+0) | 5(4+1+0) | 3(0+2+1) |
| 4 | 6(1+5+0) | 6(0+5+1) | 2(2+0+0) | 4(3+1+0) | 5(4+0+1) | 3(0+3+0) | 3(1+0+2) | 2(0+2+0) | 5(4+1+0) | 4(1+3+0) | 3(0+3+0) | 3(0+0+3) |
| 5 | 3(1+1+1) | 3(1+0+2) | 2(0+0+2) | 2(0+2+0) | 3(3+0+0) | 6(4+0+2) | 4(1+3+0) | 2(0+2+0) | 4(4+0+0) | 5(5+0+0) | 2(0+0+2) | 3(2+0+1) |
| 6 | 2(1+0+1) | 3(3+0+0) | 4(0+0+4) | 5(1+1+3) | 5(3+1+1) | 2(0+0+2) | 2(2+0+0) | 2(0+2+0) | 3(2+0+1) | 2(0+0+2) | 3(3+0+0) | 4(4+0+0) |
| 7 | 3(2+1+0) | 3(0+1+2) | 4(4+0+0) | 5(0+4+1) | 6(5+0+1) | 2(0+2+0) | 3(2+0+1) | 4(3+0+1) | 5(4+0+1) | 5(1+0+4) | 3(3+0+0) | 2(1+0+1) |
| 8 | 3(2+1+0) | 2(1+0+1) | 2(0+2+0) | 2(2+0+0) | 6(3+0+3) | 5(2+0+3) | 2(1+0+1) | 3(3+0+0) | 4(1+3+0) | 4(3+1+0) | 3(2+0+1) | 3(1+2+0) |
| 9 | 4(3+1+0) | 2(1+0+1) | 4(4+0+0) | 4(0+4+0) | 5(4+0+1) | 6(3+0+3) | 5(4+1+0) | 3(2+1+0) | 2(1+1+0) | 5(1+1+3) | 2(0+2+0) | 2(1+1+0) |
| 10 | 6(5+1+0) | 2(1+0+1) | 4(4+0+0) | 3(3+0+0) | 3(1+1+1) | 2(1+0+1) | 4(3+1+0) | 5(4+1+0) | 5(5+0+0) | 2(0+1+1) | 2(0+0+2) | 3(1+1+1) |
| 11 | 5(4+1+0) | 2(2+0+0) | 3(1+1+1) | 3(0+0+3) | 2(1+0+1) | 2(0+0+2) | 6(3+2+1) | 4(1+1+2) | 3(3+0+0) | 2(0+2+0) | 5(3+0+2) | 4(4+0+0) |
| 12 | 6(4+1+1) | 3(3+0+0) | 2(1+1+0) | 2(0+1+1) | 5(4+0+1) | 6(4+0+2) | 4(2+1+1) | 2(0+1+1) | 2(1+1+0) | 2(1+0+1) | 4(3+1+0) | 5(4+0+1) |
| 13 | 5(3+2+0) | 3(1+1+1) | 4(3+1+0) | 4(3+0+1) | 4(4+0+0) | 3(3+0+0) | 5(4+0+1) | 4(4+0+0) | 2(0+2+0) | 4(1+1+2) | 3(1+1+1) | 3(2+1+0) |
| 14 | 2(1+1+0) | 4(1+3+0) | 4(1+3+0) | 2(1+0+1) | 6(3+2+1) | 4(3+1+0) | 2(1+1+0) | 3(1+1+1) | 3(1+1+1) | 4(1+1+2) | 5(4+1+0) | 2(2+0+0) |
| 15 | 5(4+0+1) | 2(0+2+0) | 2(1+1+0) | 2(2+0+0) | 5(5+0+0) | 2(1+1+0) | 4(3+1+0) | 2(0+1+1) | 6(5+0+1) | 2(1+0+1) | 5(3+1+1) | 4(0+4+0) |



**7.3.5** Алгоритм обнаружения и исправления

**7.3.5.1** Модель, реализующая обнаружение тупиков и восстановление после них на языке GPSS/PC (файл modos75.gps), основывается на тех же данных, что и модель, реализующая предварительное распределение ресурсов (см. пункт 7.3.1).

Возможный тупик разрешается следующим образом: процесс, получивший отказ в выделении второго ресурса, добровольно покидает занимаемый им ресурс и становится в очередь к этому ресурсу снова, при этом вся работа по использованию первого ресурса (для процессов X это ресурс P1, а для Y – P2) теряется. При этом есть такой нюанс: дедлок фиксируется в том случае, если процесс (например X), захвативший несколько единиц одного типа ресурса (например типа P1) и пытающийся захватить несколько единиц ресурса другого типа (P2), "видит", что другой процесс (процесс Y) не может добиться требуемого ему ресурса (типа P1), уже владея при этом несколькими единицами ресурса типа P2. Если же процесс X, пытаясь захватить ресурс P2, "видит", что процесс Y уже владеет и использует два типа ресурсов, то дедлок не фиксируется, поскольку рано или поздно процесс Y освободит занимаемые им ресурсы обоих типов.

## 7.4 Порядок выполнения работы

**7.4.1** Получить у преподавателя номер варианта исходных данных для моделей анализа подходов к решению проблемы тупиков (номера с 1 по 12).

**7.4.2** Скопировать в свой рабочий каталог D:\OS из каталога D:\OS\LABOS\LABOS7 файлы modos71.gps,...,modos75.gps, содержащие тексты имитационных моделей, реализующих различные подходы к разрешению проблемы тупиков, а также файлы lab71.bat,...,lab75.bat и файл startup.gps для автоматизации работы с моделями систем на языке GPSS/PC. Предварительно в рабочем каталоге должны быть размещены следующие обязательные для работы файлы: gpsspc.exe, gpssrept.exe, settings.gps и position.gps (из каталога D:\OS\GPSS).

**7.4.3** Во всех GPSS-моделях изменить содержание поля операнда оператора

VARIANT VARIABLE номер_варианта

в соответствии со своим вариантом.

**7.4.4** Получить результаты моделирования с помощью имитационных моделей посредством запуска соответствующих batch-файлов, не забывая при этом модифицировать содержимое файла startup.gps. Уточненные значения средних длин очередей и среднего числа занятых единиц ресурсов и процессоров (с точностью до двух знаков после запятой) см. в соответствующих txt–файлах соответственно в колонке AVE.CONT. объектов типа QUEUE (очередь) и в колонке AVE.C. объектов типа STORAGE (накопитель). Сравнить между собой результаты выполнения моделей modos71.gps, modos72.gps и modos74.gps и сделать выводы. Величины X_P1P2 и Y_P2P1 в файле modos74.gps носят технологический характер.

**7.4.5** Построить последовательность надежных состояний по алгоритму банкира для исходных данных своего варианта (см. подпункт 7.2.3.4 и таблицу 7.3).

Примечания

1 Анонсы процессов на ресурсы P1 и P2 присутствуют в выходных результатах модели modos74.gps.

2 Поскольку процесс накапливает выделенные единицы ресурса, не освобождая их вплоть до своего завершения, то конкретные моменты времени возникновения запросов на ресурсы не принципиальны для построения возможной последовательности надежных состояний, а важна лишь последовательность запросов на количество единиц ресурсов обоих типов.

3 Новый запрос на ресурс не возникнет до тех пор, пока не будет удовлетворен предыдущий запрос.

**7.4.6** Оформить отчет и сформулировать выводы по работе (в виде текстового файла).

## 7.5 Контрольные вопросы

**7.5.1** Дайте определение понятия тупика.

**7.5.2** Приведите пример простого тупика с участием трех процессов и трех ресурсов. Проиллюстрируйте его.

**7.5.3** Сформулируйте необходимые условия возникновения тупика.

**7.5.4** Что такое "бесконечное откладывание" и чем оно отличается от тупика? что у них общего?

**7.5.5** Перечислите подходы к разрешению проблемы тупиковых ситуаций.

**7.5.6** Перечислите и раскройте смысл дисциплин предотвращения тупиковых ситуаций при распределении ресурсов.

**7.5.7** В чем смысл алгоритма банкира?

**7.5.8** Сам факт, что состояние является ненадежным, не обязательно говорит о том, что в системе возникнет тупик. Объясните, почему это так. Приведите пример ненадежного состояния и покажите, каким образом все процессы могут завершиться без тупика.

**7.5.9** В контексте алгоритма банкира определите, является ли каждое из приведенных ниже состояний надежным или ненадежным. Если состояние надежно, то покажите, каким образом могут завершиться все процессы; если состояние ненадежно, то покажите, каким образом может возникнуть тупик (таблицы 7.4 и 7.5).



Таблица 7.4 – Состояние А

| Процессы | Текущее количество выделенного ресурса | Резерв | Максимальная потребность (анонс) |
|---|---|---|---|
| Процесс 1 | 2 | | 6 |

| | | | |
|---|---|---|---|
| Процесс 2 | 4 | | 7 |
| Процесс 3 | 5 | | 6 |
| Процесс 4 | 0 | | 2 |
| Итого | 11 | 1 | – |

Таблица 7.5 – Состояние Б

| Процессы | Текущее количество выделенного ресурса | Резерв | Максимальная потребность (анонс) |
|---|---|---|---|
| Процесс 1 | 4 | | 8 |
| Процесс 2 | 3 | | 9 |
| Процесс 3 | 5 | | 8 |
| Процесс 4 | 0 | | 0 |
| Итого | 12 | 2 | – |

**7.5.10** Почему восстановление после тупиков является столь трудной проблемой? Какие способы восстановления вы знаете?

**7.5.11** Предположим, что все ресурсы идентичны, они могут захватываться и освобождаться строго по одному в каждый конкретный момент времени, причем ни одному процессу никогда не требуется больше ресурсов, чем имеется в системе. Исходя из сказанного, укажите, сможет ли возникнуть тупик в каждой из следующих систем, представленных таблицей 7.6.

Таблица 7.6

| Система | Число процессов | Число ресурсов |
|---|---|---|
| а) | 1 | 1 |
| б) | 1 | 2 |
| в) | 2 | 1 |
| г) | 2 | 2 |
| д) | 2 | 3 |

# 8 КОМАНДЫ И КОМАНДНЫЕ ФАЙЛЫ

## 8.1 Цель работы

**8.1.1** Изучить команды Windows.

**8.1.2** Приобрести навыки написания командных файлов и освоить их практическое использование.

## 8.2 Общие сведения

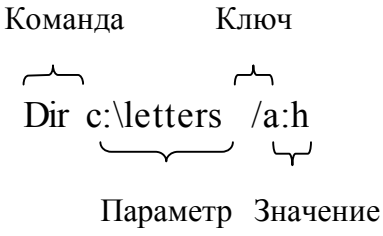
**8.2.1** Команды Windows 9x

**8.2.1.1** Все необходимые операции по работе с файлами и обслуживанию машины можно делать, не используя графический интерфейс, а используя команды. Иногда это даже проще, чем “пробиваться” через массу окон.

Любую исполнимую программу можно рассматривать как команду, где имя программы есть имя команды, которая выполняет заложенные в программе действия. Для выполнения программы часто требуются внешние данные (параметры), которые в интегрированных средах (например, в интерфейсе Windows) задаются в каком-либо окне или меню. Если эта программа запускается в командной строке, то параметры записываются вслед за именем программы. Короче, команда есть та же программа, только системная.

ОС Windows 9x полностью сохранила структуру и синтаксис команд MS–DOS. Она сохранила в основном состав команд, а также несколько расширила его, главным образом за счет сетевых команд. Команды используются для запуска утилит ОС и приложений, при написании командных файлов и т.д. [2–4,12,14,15].

**8.2.1.2** Команда может содержать до четырех элементов: *имя команды, параметры, ключи и значения* (рисунок 8.1).




По этой команде будет выведен список всех скрытых файлов в каталоге **c:\letters**. Кроме этих четырех элементов используются *командные символы,* они указывают устройство вывода информации (отличного от принимаемого по умолчанию).

*Имя команды* указывает операцию, которую будет выполнять Windows 9x.

*Параметр* указывает или создает объект, с которым будет работать Windows 9x. Параметров может быть несколько. Например, переименовать файл letter в файл memo:

**ren** letter.txt memo.txt .

Порядок следования параметров определяет работу команды, при этом первым идет параметр, определяющий *источник*, а второй – *приемник*. Иногда параметры могут разделяться точкой с запятой.

*Ключ* определяет режимы работы команды. Он определяется наклонной чертой или дефисом, за которым, как правило, следуют ключевые слова, символы или числа. Если параметров несколько, они разделяются пробелом. Ключ может стоять в любом месте после имени команды.

*Значение* определяет особенности действия ключа, обозначается двоеточием или значком равенства, за которым следует слово, символ или число. Значение указывают за ключом, не отделяя пробелом. Например:

**format d: /f:1.2 /v:backup2** .

Здесь команда **format** содержит два значения: объем диска 1,2 Мб и метку тома (backup2).

**8.2.1.3** В командах можно использовать так называемые *средства перенаправления ввода-вывода (командные символы)* (таблица 8.1).

Таблица 8.1 – Средства перенаправления ввода-вывода

| Командный символ | Действие |
|---|---|
| Команда > имя_файла | Перенаправление в файл сообщений, выводимых с помощью указанной команды (если файл уже существовал, то он заменяется новым) |
| Команда >> имя_файла | Перенаправление в файл сообщений, выводимых с помощью указанной команды (если файл уже существовал, то сообщения добавляются в конец этого файла) |
| Команда < имя_файла | Чтение входных данных команды не с клавиатуры, а из файла |
| Команда \| команда | Передача сообщений, выводимых на экран первой командой, в качестве входных данных для второй команды |

Наиболее часто они используются в командах **sort**, **more**, **find**, которые позволяют сортировать вводимые и выводимые данные, выдавать результаты поэкранно, а также искать в файле определенный текст. Например, команда

**dir | more**

поэкранно выведет содержание текущей папки.

**8.2.1.4** Для вывода на экран подсказки по конкретной команде необходимо в командной строке ввести нужную команду и через пробел – **/?**. Например,

**dir /?** .

**8.2.1.5** Команды можно вводить в командной строке сеанса MS–DOS. В пункте **Выполнить** Главного меню Windows выполняются только внешние команды.

**8.2.1.6** Чтобы приостановить выполнение команды, нужно нажать **Ctrl+C** или **Pause**. Возобновляется действие команды нажатием любой клавиши, кроме **Pause**.

**8.2.1.7** Чтобы остановить выполнение команды**,** надо нажать **Ctrl+Break** или **Ctrl+C**.

**8.2.1.8** Сеанс MS–DOS может выполняться в оконном или полноэкранном режимах. В первом случае инструменты окна действуют без ограничений (можно переносить текст через буфер обмена и т.д.). Переключаться между режимами можно нажатием **Alt+Enter**.

**8.2.1.9** Команды бывают *внутренние* и *внешние*. *Внутренние* постоянно присутствуют в памяти, а *внешние* хранятся в виде файлов и при вызове загружаются с диска (таблица 8.2). Примеры команд, используемых только в командных файлах, будут даны ниже.

Таблица 8.2 – Список некоторых команд

| Команда | Признак | Описание |
|---|---|---|
| ATTRIB | Внешняя | Показывает или меняет атрибуты файла |
| CD (CHDIR) | Внутренняя | Вывод имени либо смена текущей папки |
| CHKDSK | Внешняя | Проверка диска и вывод статистики |
| CLS | Внутренняя | Очистка экрана |
| COMMAND | Внешняя | Запуск новой копии интерпретатора команд Windows |
| COPY | Внутренняя | Копирование одного или нескольких файлов в другое место |
| DATE | Внутренняя | Вывод либо установка текущей даты |
| DEL (ERASE) | Внутренняя | Удаление одного или нескольких файлов |
| DELTREE | Внешняя | Удаление папки вместе с подпапками и содержащимися в них файлами |
| DIR | Внутренняя | Вывод списка файлов и подпапок из указанной папки |

Продолжение таблицы 8.2

| Команда | Признак | Описание |
|---|---|---|
| FIND | Внешняя | Поиск текстовой строки в одном или нескольких файлах |
| FORMAT | Внешняя | Форматирование диска для работы с MS-DOS |
| RD (RMDIR) | Внутренняя | Удаление пустой папки |
| MEM | Внешняя | Вывод сведений о полной и свободной системной памяти |
| MORE | Внешняя | Последовательный вывод данных по частям размером в один экран |
| MOVE | Внешняя | Перемещение одного или более файлов |
| PATH | Внутренняя | Вывод либо установка пути поиска исполняемых файлов |
| PAUSE | Внутренняя | Приостановка выполнения пакетного файла и вывод сообщения: Нажмите любую клавишу.... |
| SET | Внутренняя | Вывод, установка и удаление переменных среды Windows |
| SORT | Внешняя | Сортировка ввода с выводом результатов в файл, на экран или другое устройство |
| TIME | Внутренняя | Вывод и установка системного времени |
| TYPE | Внутренняя | Вывод на экран содержимого текстовых файлов |
| XCOPY | Внешняя | Копирует файлы и структуру папок |

**8.2.2** Командные файлы

**8.2.2.1** *Командным* или *пакетным* (**batch files** - пакетные файлы) файлом (КФ) называется последовательность команд Windows, записанная в текстовый файл и выполняемая путем задания имени этого файла аналогично исполняемой команде. Такой файл представляет собой системную макрокоманду и является аналогом процедуры в программах.

**8.2.2.2** КФ создаются любым текстовым редактором, которые формируют файлы типа **.txt**, или командой:

**copy con имя_файла** ,

где **con** — зарезервированное имя для клавиатуры.

**8.2.2.3** КФ должны иметь расширение **bat**.

**8.2.2.4** КФ предназначены для задания часто используемых последовательностей команд. Они могут содержать любые команды, допустимые в командной строке. Кроме того, имеются дополнительные команды, которые

используются только в таких файлах. По существу, КФ – та же программа на своем языке, как, например, на Паскале, где имя процедуры соответствует имени КФ, а ее операторы – командам в КФ.

**8.2.2.5** Командные файлы обрабатываются построчно, а прервать его выполнение можно командой **Ctrl+C** или **Ctrl+Break**.

**8.2.2.6** Разрешено из одного командного файла осуществлять вызов другого командного файла с последующим продолжением работы первого файла.

**8.2.2.7** Среди командных файлов есть файл с зарезервированным именем **autoexec.bat**. Он отличается от других только тем, что помещается в корневой каталог системного диска и содержит команды, которые пользователь хочет ввести при загрузке ОС. В случае отсутствия такого файла все установки определяются по умолчанию.

**8.2.2.8** Командный файл можно вызвать в пошаговом режиме. Это может быть удобно для отладки командных файлов. Формат команды:

**command** /y /c имя_командного_файла [параметры] .

Текст каждой команды будет выводиться перед выполнением на экран. Для выполнения команды надо нажать клавишу **Y** или **Enter**, для пропуска команды – **N** или **Esc**.

**8.2.3** Параметры командных файлов

**8.2.3.1** Командным файлам из командной строки могут быть переданы любые аргументы. Аргументы задаются параметрами командной строки после имени файла. Например, файл **delbak.bat**, удаляющий файлы с расширением **.bak** и принимающий параметры – пути, по которым необходимо произвести удаление, можно выполнить командой:

**delbak d:\ e:\ e:\stud e:\work f:** .

**8.2.3.2** Количество аргументов командного файла ограничено лишь размером командной строки. Однако напрямую командный файл может обработать лишь девять параметров. Для доступа к аргументам в файле используется своего рода макроподстановка. Параметры в файле имеют имена, обозначаемые символами %1 – %9. В командном файле можно использовать также символ %0, значение которого - имя самого командного файла (в той форме, в которой оно указано в команде, вызвавшей командный файл). Если при вызове командного файла задано меньше девяти параметров, то «лишние» символы из %1–%9 замещаются пустыми строками. Для доступа к параметрам, следующим за девятым, используется команда **shift**. Эта команда сдвигает аргументы командного файла на один влево, таким образом, после выполнения команды **shift** ко второму аргументу командного файла можно обратиться по имени %1. Команда **shift** работает только в одну сторону, поэтому сдвинутые влево аргументы теряются. Команду **shift** можно использовать несколько раз.

**8.2.3.3** Если в командном файле знак процента используется не для обозначения параметров, а для других целей, то его надо набрать дважды. Например,

чтобы в командном файле указать файл **xyz%.com**, надо написать в строке командного файла **xyz%%.com**.

**8.2.4** Команды для командных файлов

**8.2.4.1** По умолчанию команды пакетного файла выводятся на экран перед выполнением. Если в пакетный файл вставить команду **echo off**, то выполняемые за ней команды не будут выводиться на экран. А команда **echo on** включает режим вывода выполняемых команд на экран. Можно избежать вывода (дублирования) на экран и любой отдельной строки командного файла. Для этого надо поставить в начале этой строки командный префикс **@**.

Примечания

1 Обычно в качестве первой строки командного файла используется команда **@echo off**. При этом строки командного файла на экран не выводятся.

2 После выдачи команды **echo off** может быть полезно использовать команду **cls** для более удобного просмотра сообщений, выводимых из командного файла.

**8.2.4.2** Команда **echo** позволяет выдавать из командного файла сообщения на экран. Формат команды:

**echo** сообщение .

Сообщение выдается независимо от установки **on** или **off**, при этом сообщение не может быть пустым или равным **on** или **off**. Символы «<», «>» и «|» в сообщении недопустимы. Перед командой **echo сообщение** желательно выполнить команду **@echo off**, чтобы сообщение не выводилось на экран дважды. Для вывода пустой строки можно воспользоваться командой **echo.** (точка должна следовать сразу за словом «**echo**»).

С помощью средств перенаправления ввода-вывода DOS (см. таблицу 8.1) можно выводить сообщения не на экран, а в файл. Формат команды:

– для добавления строки с сообщением в конец файла (если файл не существует, то он создается)

**echo** сообщение >> имя_файла ;

– для создания файла и записи в него строки с сообщением (если такой файл уже существует, то его старое содержимое будет потеряно)

**echo** сообщение > имя_файла .

**8.2.4.3** Для приостановки выполнения командного файла можно использовать команду **pause**. Формат команды:

**pause** [сообщение] .

Аргумент «сообщение» выдается только при **echo on**. При выполнении команды **pause** на экран выводится строка **Strike a key when ready...** (Нажмите любую клавишу, когда будете готовы) и выполнение командного файла приостанавливается. Если нажать **Ctrl+C** или **Ctrl+Break**, то выполнение командного файла можно либо закончить (ответ **Y**), либо продолжить со следующей команды (ответ **N**).

Иногда бывает полезно убрать данное сообщение ОС. Это можно сделать, переназначив вывод команды **pause** на пустое логическое устройство **nul**, например:

**pause > nul** .

**8.2.4.4** Команда **rem** позволяет включать в командный файл комментарии, которые не будут интерпретироваться как команды во время исполнения этого файла. Формат команды:

**rem** комментарий .

Примечания

1 В комментарии не следует употреблять символы «<», «>» и «|» – они интерпретируются как символы перенаправления ввода-вывода.

2 Перед комментариями, которые нежелательно выводить на экран даже при отладке (т.е. в режиме **echo on**), целесообразно ставить символ "***@***".

**8.2.4.5** Если из командного файла вызвать другой командный файл, вставив в него имя этого командного файла с необходимыми параметрами, то после завершения вызванного файла возврата управления в исходный командный файл не произойдет. Если же такой возврат необходим, то следует использовать команду **call**. Формат команды:

**call** имя_командного_файла [параметры] .

Если в командной строке указаны какие-либо параметры, кроме имени командного файла, то эти параметры передаются командному файлу, они доступны там как значения символов %1–%9. По окончании выполнения вызванного командного файла продолжается (со следующей строки) выполнение исходного командного файла. Заметим, что в команде **call** не допускается перенаправление ввода–вывода. Допускается создавать рекурсивные командные файлы.

**8.2.4.6** Командный файл может содержать метки и команды перехода. Это позволяет управлять порядком выполнения команд в файле.

Любая строка командного файла, начинающаяся с двоеточия «**:**», воспринимается при обработке командного файла как метка. Имя метки определяется набором символов, следующих за двоеточием до первого пробела или конца строки (остаток строки после первого пробела игнорируется и воспринимается как комментарий).

Чтобы выполнение команд в командном файле было продолжено со строки, которая следует сразу после некоторой метки, надо воспользоваться командой

**goto** [:]метка .

Если метка в команде **goto** не указана или не найдена в командном файле, то выполнение командного файла завершается.

**8.2.4.7** Команда **if** позволяет в зависимости от выполнения некоторых условий выполнять или не выполнять команды в командном файле. Формат команды:

**if** условие команда .

Параметры данной команды следующие:

– команда – это любая допустимая команда (в том числе **goto**). Эта команда выполняется, если условие в команде **if** истинно, в противном случае команда игнорируется;

– условие – это одно из приведенных ниже выражений:

1) **errorlevel** n – условие истинно тогда, когда код завершения предыдущей выполненной программы не меньше, чем целое число n (код завершения устанавливается программами при окончании их работы, по умолчанию этот код равен нулю);

2) строка1 == строка2 – условие истинно, если строка1 и строка2 полностью совпадают. Если в этих строках имеются символы %0–%9, то вместо этих символов подставляются параметры командного файла;

3) **exist** имя_файла – условие истинно тогда, когда указанный файл существует;

4) **not** условие – истинно тогда, когда указанное условие ложно.

Примечания

1 Команды **if** могут быть вложенными.

2 Если в качестве одной из строк конструкции «строка1==строка2» требуется задать пустую строку, то следует к обеим строкам добавить одни и те же символы, например: .стр1.==.., "стр1"==="", !стр1==!.

3 В связи с тем, что код завершения анализируется не на равенство, а на больше либо равно, для правильной работы команды **if** с условием **errorlevel** следует использовать схему вида:

```
…
if errorlevel n goto меткаN
…
if errorlevel 2 goto метка2
if errorlevel 1 goto метка1
…
```

**8.2.4.8** Иногда в командном файле нужно выполнить различные действия по выбору пользователя. Это можно сделать с помощью программы **choice**. Формат команды:

**choice** [/C[:]список_символов] [/N] [/S] [/T[:]символ,число_секунд] сообщение .

Данная команда отображает приглашение и ждет нажатия клавиши, устанавливая переменную среды (системная переменная) **errorlevel** в значение, соответствующее порядковому номеру символа, заданного в списке. Параметры команды следующие:

– /C[:]список_символов – указывает допустимые символы, которые может ввести пользователь в ответ на сообщение, при этом значение переменной **errorlevel** устанавливается равным номеру введенного в списке символа (символы в списке идут подряд без всяких разделителей). Если данный параметр не указан, то допустимые символы – это **Y** и **N**;

– /T[:]символ,число_секунд – если этот параметр указан, то в случае, когда пользователь по истечении заданного параметром «число_секунд» времени не нажал ни на одну клавишу, принимается ответ «символ». Символ должен быть представлен в списке в ключе /C. Секунды могут быть заданы в диапазоне от 1 до 99 (если 0, то ожидание бесконечно);

– сообщение – указывает сообщение, выводимое на экран.

Примечания

1 Если вы желаете, чтобы при вводе символов различались прописные и строчные буквы, укажите в команде параметр /S.

2 Обычно к сообщению добавляется список допустимых для ответа символов (через запятую в квадратных скобках) и знак вопроса. Если вы не хотите, чтобы к сообщению выводился такой «довесок», укажите в команде параметр /N.

**8.2.4.9** Команда **for** предназначена для организации цикла в командном файле. Формат команды:

**for** %%x **in** (список) **do** команда .

Параметры команды следующие:

– x – любой символ (кроме цифр и некоторых специальных символов); обычно это буква;

– список – список значений переменной, которые она принимает во время работы цикла (например, одно или несколько имен файлов, разделенных пробелами);

– команда – любая программа или команда, кроме команды **for**; выполняется столько раз, сколько параметров присутствует в списке (как правило, команда содержит имя переменной x). Если необходимо выполнить несколько команд, то следует записать их в отдельный командный файл и использовать для его вызова команду **call** (см. подпункт 8.2.4.5).

Примечания

1 Команды **for** не могут быть вложенными

2 Команда **for** может использоваться и вне командного файла, но в этом случае параметр цикла должен начинаться с одного символа %.

**8.2.4.10** В командных файлах можно использовать значения переменных окружения. Для установки переменных окружения служит команда **set**. Формат команды:

**set** переменная=[значение] ,

где переменная – любая строка, не содержащая знаков равенства и пробелов. При этом в переменной большие и малые латинские буквы считаются одинаковыми;

значение – любая строка символов.

Команда **set** записывает строку “переменная=значение” в специальную область памяти, зарезервированную для хранения переменных окружения. Если переменной уже было присвоено какое-либо значение, то оно заменяется новым. Если значение – пустая строка, то строка, задающая значение переменной, удаляется из области памяти, зарезервированной для хранения переменных окружения.

Если в командном файле употребить имя переменной окружения (в том числе и глобальной), заключенное с обеих сторон в знаки процента, то оно будет замещено на значение этой переменной. Например, после ввода команды **set** chifiles=c:\chi строка %chifiles% в командном файле будет замещена на c:\chi. Или, например, командный файл, добавляющий новый маршрут поиска исполняемых файлов к уже имеющимся маршрутам (значение глобальной переменной окружения path), можно представить строкой вида

**set** path=%path%;%1.

**8.2.5** Примеры командных файлов

**8.2.5.1** Пример использования команды **for** в командном файле:

```
@echo off
for %%a in (работает цикл for) do echo %%a .
```

Файл, состоящий из этих двух строк, выведет на экран текст:

```
работает
цикл
for
```

**8.2.5.2** Командный файл **delbak.bat** для удаления файлов с расширением bak по указанным в параметрах файла путям:

```
@echo off
:Clear
shift
if .%0. == .. goto End_of_file > nul
if exist %0*.bak del %0*.bak
goto Clear
:End_of_file
```

**8.2.5.3** Командный файл **typ.bat** для вывода некоторого файла на экран (имя выводимого файла указывается в качестве параметра):

```
@echo off
if –%1 == – goto no_param
if not exist %1 goto not_exist
type %1 | more
goto exit
:no_param
echo Должен быть задан параметр (файл для отображения)
goto exit
:not_exist
echo Файл %1 не найден
:exit
```

## 8.3 Порядок выполнения работы

**8.3.1** Ознакомиться с описанием лабораторной работы.

**8.3.2** Выполнить одно из нижеперечисленных заданий (по указанию преподавателя). Все индивидуальные задания включают в себя подготовку командного файла, реализованного по стандартным правилам написания программ, т.е. он должен реагировать на возможные ошибки, избегая сообщений ОС, выдавать сообщения о ходе работы и промежуточную информацию, в том числе и справочную. Темы индивидуальных заданий, с согласия преподавателя, могут быть предложены и самими студентами.

**8.3.3** Темы индивидуальных заданий

**8.3.3.1** Обеспечить слияние n (значение n должно быть произвольным) файлов в один с удалением исходных; файлы задаются параметрами командной строки.

**8.3.3.2** Построить меню с тремя альтернативами, обеспечивающее выполнение одной из трех программ; предусмотреть выбор одного из пунктов меню по умолчанию и выход из командного файла без выбора программ.

**8.3.3.3** Первый параметр командного файла содержит путь, по которому создается папка (директорий, каталог) с именем, заданным вторым параметром, и в нее переносятся файлы, список которых представлен остальными параметрами.

**8.3.3.4** Обеспечить обмен файлов между двумя указанными папками (папки задаются в параметрах командного файла).

**8.3.3.5** Вывести на экран с помощью командного файла свою фамилию (параметр 2), имя (параметр 3) и отчество (параметр 4) по паролю (параметр 1).

**8.3.3.6** Обеспечить с помощью командного файла запись фамилии (параметр 1, фамилию набирать в алфавите клерного письма) в файл (параметр 2) с автоматической сортировкой от A до Z.

**8.3.3.7** Создать командный файл, позволяющий по значению /w (параметр 1) записывать в телефонную книжку (файл) Ф.И.О. (параметр 2) и соответствующий номер телефона (параметр 3), а по значению /r (параметр 1) – узнавать номер телефона по Ф.И.О. (параметр 2).

**8.3.3.8** Построить командный файл для удаления заданных файлов, запрещающий удалять файлы типа exe, com и bat, а также запрашивающий подтверждение на удаление и предусматривающий восстановление нечаянно удаленных файлов. После своей работы он не должен оставлять никаких “следов”.

**8.3.3.9** Построить командный файл для получения перемещаемого изображения, пользуясь лишь командами MS–DOS.

**8.3.4** Убедиться в правильности функционирования созданного КФ.

**8.3.5** Продемонстрировать преподавателю работу созданного командного файла.

## 8.4 Контрольные вопросы

**8.4.1** Структура команды (ее элементы).

**8.4.2** Порядок следования параметров в команде.

**8.4.3** Как вывести подсказку на команду.

**8.4.4** Как приостановить и как прервать выполнение команды.

**8.4.5** Чем отличаются внутренние и внешние команды.

**8.4.6** Дать определение командному файлу, его назначению.

**8.4.7** Какие команды можно использовать в командных файлах?

**8.4.8** Каким образом командный файл может получить информацию извне?

**8.4.9** Как из одного командного файла выполнить другой?

**8.4.10** Какие вы знаете команды специально (или преимущественно) для употребления в командных файлах?

**8.4.11** Как создать командный файл.

# ЛИТЕРАТУРА


1. Альянах Н. Моделирование вычислительных систем. – Л.: Машиностроение. Ленингр. отд-ние, 1988.– 223 с.

2. Богумирский Б.С. Руководство пользователя ПЭВМ: В 2–х ч. Ч.1. – СПб: Ассоциация ОШСО, 1992. – 357 с.

3. Богумирский Б.С. Руководство пользователя ПЭВМ: В 2–х ч. Ч.2. – СПб: Ассоциация ОШСО, 1992. – 378 с.

4. Богумирский Б.С. Эффективная работа на IBM PC в среде Windows 95. – СПб: Питер, 1997. – 1120 с.

5. Дейтел Г. Введение в операционные системы: В 2-х т. Т.1. Пер. с англ.– М.: Мир, 1987.– 359 с.

6. Дейтел Г. Введение в операционные системы: В 2-х т. Т.2. Пер. с англ.– М.: Мир, 1987.– 398 с.

7. Иванчиков А.А., Ревотюк М.П. Лабораторный практикум по курсу “Системное программирование” для студентов специальности “Автоматизированные системы обработки информации и управления“. – Мн.: БГУИР, 1994. – 193 с.

8. Иртегов Д.В. Введение в операционные системы. – СПб: БХВ-Петербург, 2002.–624 с.

9. Кейлингерт П. Элементы операционных систем. Введение для пользователей: Пер. с англ.– М.: Мир, 1985.–295 с.

10. Краковяк С. Основы организации и функционирования ОС ЭВМ: Пер. с франц. – М.: Мир, 1988.– 480 с.

11. Лорин Г., Дейтел Х.М. Операционные системы/Пер. с англ.; Предисл. Л.Д.Райкова. –М.: Финансы и статистика, 1984.– 392 с.

12. Нортон П., Мюллер Дж. Windows 98: Пер. с англ. – СПб: БХВ–Петербург, 2001. – 592 с.

13. Сетевые операционные системы/В.Г.Олифер, Н.А.Олифер. – СПб: Питер, 2001.– 544 с.

14. Тулинов Е.С., Хижняк А.В. Справочник пользователя IBM PC. – Мн.: Беларусь, 1999. – 525 с.

15. Фигурнов В.Э. IBM PC для пользователя. Краткий курс. – М.: ИНФРА–М, 1998. – 480 с.

16. Юлин В.А., Булатова И.Р. Приглашение к Си. – Мн.: Выш. шк., 1990. – 224 с.

Св. план 2002, поз.34

Учебное издание

Севернёв Александр Михайлович

# ОПЕРАЦИОННЫЕ СИСТЕМЫ

Методическое пособие для выполнения лабораторных работ
по курсу “Операционные системы”

для студентов специальности 53 01 02
“Автоматизированные системы обработки информации и управления”



Редактор Т.А.Лейко
Корректор Е.Н.Батурчик

Подписано в печать Формат 60x84 1/16
Бумага Печать Гарнитура Times Усл. печ. л.
Уч.-изд.л. 5,5 Тираж 150 экз. Заказ 450

Издатель и полиграфическое исполнение:
Учреждение образования
“Белорусский государственный университет информатики и радиоэлектроники”
Лицензия ЛП №156 от 05.02.2001.
Лицензия ЛВ №509 от 03.08.2001.
220013, Минск, П. Бровки, 6.