Министерство образования и науки
Российской Федерации

Государственное образовательное учреждение
высшего профессионального образования
«Московский государственный университет леса»

А. В. Чернышов

ПЛАНИРОВАНИЕ ПРОЦЕССОВ

Практикум
к выполнению лабораторных работ
Для студентов специальности 220100

Часть 1

Издательство Московского государственного университета леса
Москва — 2004

УДК 681.3

6Л2 Чернышов А. В. Планирование процессов: Практикум к выполнению лабораторных работ для студентов специальности 220100. Ч. 1. — М.: МГУЛ, 2004. — 24 с.

В учебно-методическом пособии даны необходимый минимум информации для выполнения лабораторных работ первого семестра по дисциплине «Операционные системы», а также индивидуальные задания на каждую лабораторную работу и требования к оформлению отчётов по ним.

Разработано в соответствии с Государственным образовательным стандартом ВПО 2000 г. для подготовки специалистов по направлению 654600 на основе примерной программы дисциплины «Операционные системы» для специальности 552800 (220100) «Информатика и вычислительная техника».

Одобрено и рекомендовано к изданию в качестве практикума редакционно-издательским советом университета

Рецензент — доцент Ю. А. Федотов.

Кафедра вычислительной техники

Автор — Александр Викторович Чернышов, доцент

Редактор Е. Г. Петрова

По тематическому плану внутривузовских изданий учебной литературы на 2004 г., поз. 76

Оригинал-макет выполнен в пакете teTEX с использованием кириллических шрифтов семейства LH.

Вёрстка в TEXe: А. В. Чернышов

Лицензия ЛР № 020718 от 02.02.1998 г.

Лицензия ПД № 00326 от 14.02.2000 г.

Подписано к печати		Формат 60×88/16
Бумага 80 г/см^2 «Снегурочка»		Ризография
Объем 1,5 п. л.	Тираж 100 экз.	Зак. №

Издательство Московского государственного университета леса.
141005. Мытищи-5, Московская обл., 1-я Институтская, 1, МГУЛ.
Телефон: (095) 588-57-62
e-mail: izdat@mgul.ac.ru

Предисловие

Настоящий практикум включает три лабораторные работы, которые должны быть выполнены студентами в течение первого семестра изучения дисциплины «Операционные системы», занимающей два учебных семестра.

Для выполнения работ студентам необходимо знание языка программирования Ассемблер ПЭВМ IBM PC. Предполагается, что студенты изучают этот язык в рамках специального курса.

Лабораторные работы построены таким образом, что каждая следующая работа является продолжением и расширением предыдущей, хотя иногда и не использует программный код предыдущей работы буквально.

К каждой лабораторной работе даётся необходимый минимум теоретического материала, позволяющий студентам уверенно выполнить первые две работы и получить общее представление о выполнении третьей.

Третья лабораторная работа является своего рода небольшим исследованием и потребует от каждого студента не слепого следования инструкциям, а проявления своих инженерных творческих способностей. Выполняя эту работу, студенты должны будут самостоятельно и на практике получить представление о функционировании одного из важнейших компонентов современных операционных систем — планировщика процессов.

К каждой лабораторной работе для студентов одной группы подобраны по возможности индивидуальные задания. Однако все задания сформулированы не детально, что даёт возможность каждому студенту при написании конкретных программ проявить свою индивидуальность.

Лабораторная работа 1

СРАВНЕНИЕ РЕАЛЬНОЙ И РАСШИРЕННОЙ МАШИН

1. Общие сведения

При работе с операционными системами необходимо различать понятия реальной и расширенной машины. **Реальная машина** — это набор аппаратных средств самой ЭВМ, предоставляющий, в частности, набор команд процессора, регистров ввода-вывода периферийных устройств и т. п. **Расширенная машина** — это набор стандартных подпрограмм, предоставляемых операционной системой прикладным программам в качестве стандартных средств по выполнению различных системных функций (распределение памяти, организация ввода-вывода и др.). При этом средства расширенной машины значительно проще в использовании и менее подвержены ошибкам, поскольку в них учтено множество нюансов выполнения соответствующих операций.

В качестве конкретного примера реальной и расширенной машин рассмотрим подсистему вывода текстовой информации на терминал ПЭВМ IBM PC. Здесь реальная машина представлена аппаратурой видеоадаптера, имеющего свои регистры управления режимами работы и собственную память для хранения данных. Расширенная машина представлена прерываниями BIOS и DOS.

1) Описание аппаратных средств видеоадаптера

Управление режимами видеоадаптеров — весьма сложный процесс. Типичный видеоадаптер имеет несколько десятков управляющих регистров различного назначения. Причём ошибочная запись в некоторые из них может привести к порче монитора. В печати вышло довольно большое количество литературы, посвящённой программированию видеоадаптеров. В качестве примера можно указать книгу [1].

Здесь будет рассмотрен только цветной текстовый видеорежим разрешением 80×25 как наиболее совместимый для большинства видеоадаптеров. Как правило, этот режим устанавливается по умолчанию при загрузке компьютера. Поэтому специально программировать его нет смысла.

В этом режиме информация, отображаемая на экране терминала, размещается в видеопамяти, начиная с адреса B800:0000h. Каждый отображаемый на терминале символ занимает два последовательных байта памяти: младший — код символа; старший — цвет символа. При этом младшая тетрада задаёт цвет собственно символа (часто называемый цветом переднего плана); старшая — цвет фона. Цвет кодируется по системе RGB следующим образом: самый младший бит — синий; следующий — зелёный; далее — красный. Самый старший бит в тетраде задаёт интенсивность цвета (обычный или яркий). При этом в тетраде, отвечающей за цвет фона, этот бит в зависимости от режима работы адаптера может управлять не интенсивностью, а миганием символа.

Предположим, что нам необходимо записать символ в шестую позицию пятой строки. Для этого необходимо вычислить адрес ячейки в видеопамяти, в которой должен располагаться символ для отображения в этой позиции.

Одна строка терминала занимает в видеопамяти

$$80 \text{ символов} \times 2 \text{ байта} = 160 \text{ байт}.$$

Следовательно, полное смещение от начала видеопамяти для нашего случая будет

$$160 \times (5 - 1) + 2 \times (6 - 1) = 650 = 28\text{Ah}.$$

То есть нам необходимо записать код символа по адресу B800:028Ah и код его цвета по адресу B800:028Bh.

Для вывода строки необходимо аналогичным образом заполнять последовательные байты видеопамяти. Если при этом не контролировать положение правой границы строки на терминале, то длинная строка может автоматически занять несколько терминальных строк.

2) Описание функций BIOS

Всё управление видеосервисом сведено в прерываение 10h. Здесь приводится описание только тех функций прерывания, которые могут понадобиться для выполнения лабораторной работы.

На первый взгляд может показаться, что использование функций BIOS более громоздко, чем программирование видеоадаптера напрямую. Однако использование BIOS обеспечивает значительно бóльшую

переносимость программы между различными типами видеоадаптеров. Попытка достичь такой же переносимости прямым программированием видеоадаптера потребует весьма большого объёма кода и при этом не гарантирует результата.

Функция 02h — установка позиции курсора. Применяется для позиционирования курсора. Позиция задаётся относительно верхнего левого угла экрана, имеющего координаты (0,0).

Вход:
AH=02h
BH=номер страницы (обычно 0)
DH=номер строки
DL=номер столбца
Возврат: нет.

Функция 06h — скроллинг окна вверх. Выполняет «прокрутку» содержимого заданного окна экрана вверх на заданное число строк.

Вход:
AH=06h
AL=число строк для скроллинга
BH=атрибут для вставленных снизу строк
CH=номер строки верхнего левого угла окна
CL=номер столбца верхнего левого угла окна
DH=номер строки нижнего правого угла окна
DL=номер столбца нижнего правого угла окна
Возврат: нет.

Функция 07h — скроллинг окна вниз. Выполняет «прокрутку» содержимого заданного окна экрана вниз на заданное число строк.

Вход:
AH=07h
AL=число строк для скроллинга
BH=атрибут для вставленных сверху строк
CH=номер строки верхнего левого угла окна
CL=номер столбца верхнего левого угла окна
DH=номер строки нижнего правого угла окна
DL=номер столбца нижнего правого угла окна
Возврат: нет.

Функция 08h — чтение символа и атрибута в позиции курсора. Читает символ и атрибут в текущей позиции курсора. Курсор не перемещается. Возвращает код символа в регистре AL и атрибут в регистре AH.

Вход:
AH=08h
BH=страница (обычно 0)
Возврат:
AL=код символа
AH=атрибут символа.

Функция 09h — запись символа и атрибута в позицию курсора. Записывает символ и атрибут заданное число раз, начиная с позиции курсора. Курсор не перемещается. Специальные символы (перевод строки и т. п.) отдельно не выделяются и выводятся, как и другие символы.

Вход:
AH=09h
AL=код символа
BH=страница (обычно 0)
BL=цвет символа и фона (атрибут)
CX=коэффициент повторения
Возврат: нет.

Функция 0Ah — запись символа в позицию курсора. Записывает символ заданное число раз, начиная с позиции курсора. Цветовой атрибут не изменяется. Курсор не перемещается. Специальные символы (перевод строки и т. п.) отдельно не выделяются и выводятся, как и другие символы.

Вход:
AH=0Ah
AL=код символа
BH=страница (обычно 0)
CX=коэффициент повторения
Возврат: нет.

Функция 0Eh — запись символа в позицию курсора в телетайпном режиме. Записывает символ в позицию курсора. Цветовой атрибут не изменяется. Курсор перемещается в следующую позицию. Распознаются и обрабатываются четыре управляющих символа: звонок — 07h; возврат на шаг — 08h; перевод строки — 0Ah; возврат каретки — 0Dh.

Вход:
AH=0Eh
AL=код символа
BH=страница (обычно 0)

Возврат: нет.

Функция 13h — запись цепочки символов (строки). Записывает в видеобуфер цепочку символов с указанными атрибутами. Распознаёт и обрабатывает четыре указанных выше управляющих символа.

Если атрибуты ожидаются в самой цепочке символов, то за каждым кодом символа должен следовать код его атрибута.

Вход:

AH=13h

AL=

00h — в BL атрибут, курсор не перемещается

01h — в BL атрибут, курсор перемещается

02h — атрибуты в цепочке, курсор не перемещается

03h — атрибуты в цепочке, курсор перемещается

BH=страница (обычно 0)

BL=атрибут

CX=длина цепочки

DH=номер строки экрана

DL=номер столбца экрана

ES:BP=начальный адрес цепочки

Возврат: нет.

3) Описание прерываний DOS

Описываемый сервис DOS ориентирован на вывод данных в стандартный поток вывода, который по умолчанию связан с экраном терминала. В связи с этим появляется возможность перенаправить весь вывод на другое устройство или в файл, практически не изменяя самой программы. Весь сервис представлен функциями прерывания 21h. Обратите внимание, что в сервисе DOS не предусмотрена работа с цветовыми атрибутами.

Функция 02h — выдать символ на стандартный вывод. Обрабатывает символ 08h (шаг назад). Распознаёт Ctrl Break и вызывает обработчик.

Вход:

AH=02h

DL=код символа

Возврат: нет.

Функция 06h* — выдать символ на стандартный вывод.

Вход:

AH=06h

DL=код символа от 0h до 0FEh

Возврат: нет.

Функция 09h — выдать строку на стандартный вывод. Строка должна быть ограничена символом '$' (24h). Распознаются и обрабатываются символы 08h (шаг назад), 0Ah (перевод строки), 0Dh (возврат каретки). Кроме того, опознаётся Ctrl Break и передаётся управление обработчику.

Вход:

AH=09h

DS:DX=адрес начала выводимой строки. Строка должна завершаться символом '$'.

Возврат: нет.

Функция 40h* — выдать строку на стандартный вывод. Осуществляет выдачу строки на стандартный вывод как запись в стандартный файл вывода. Строка ограничивается по длине.

Вход:

AH=40h

BX=описатель файла (в данном случае 1)

DS:DX=адрес начала выводимой строки

CX=число байт

Возврат:

AX=код ошибки, если установлен CF

AL=число реально выданных байт

2. Задание на лабораторную работу

Написать на ассемблере IBM PC программу вывода строки на экран:

– прямой записью в видеопамять;

– с использованием функции BIOS.

При этом руководствоваться требованиями к программе, представленными в табл. 1.1.

---

* Помеченные таким знаком функции имеют более широкое назначение, чем описанное здесь.

Т а б л и ц а 1.1

Номер в журнале	Цветность символов	В цветности задан	Направление строки
1	Один цвет	Цвет символов	↑
2	Разные цвета	Цвет фона	↑
3	Один цвет	Оба цвета	→
4	Разные цвета	Цвет символов	→
5	Один цвет	Цвет фона	↓
6	Разные цвета	Оба цвета	↓
7	Один цвет	Цвет символов	←
8	Разные цвета	Цвет фона	←
9	Один цвет	Оба цвета	↑
10	Разные цвета	Цвет символов	↑
11	Один цвет	Цвет фона	→
12	Разные цвета	Оба цвета	→
13	Один цвет	Цвет символов	↓
14	Разные цвета	Цвет фона	↓
15	Один цвет	Оба цвета	←
16	Разные цвета	Цвет символов	←
17	Один цвет	Цвет фона	↑
18	Разные цвета	Оба цвета	↑
19	Один цвет	Цвет символов	→
20	Разные цвета	Цвет фона	→
21	Один цвет	Оба цвета	↓
22	Разные цвета	Цвет символов	↓
23	Один цвет	Цвет фона	←
24	Разные цвета	Оба цвета	←

3. Содержание отчёта

Отчёт должен содержать:

– титульный лист;

– задание на работу;

– краткое описание алгоритма работы с аппаратурой, а также используемых прерываний и функций;

– тексты программ.

4. Навыки, полученные студентом

После выполнения лабораторной работы студент должен:

– иметь представление о понятиях реальной и расширенной машин;

– знать назначение расширенной машины IBM PC;

– владеть методами проектирования расширенной машины на базе реальной;

– уметь применять сервис расширенной машины IBM PC.

Лабораторная работа 2

ОБРАБОТКА АППАРАТНОГО ПРЕРЫВАНИЯ

1. Общие сведения

Понятие прерывания является одним из важнейших в мире многозадачных операционных систем. Различные периферийные устройства время от времени требуют обслуживания центральным процессором. Моменты времени, когда это обслуживание может потребоваться, заранее неизвестны. Но чем точнее совпадёт момент необходимости обслуживания с готовностью процессора выполнить это обслуживание, тем эффективнее будет работать вся система.

Многие устройства для уведомления процессора о необходимости их обслуживания генерируют аппаратные прерывания. В ПЭВМ типа IBM PC это таймер, клавиатура, часы реального времени, порт принтера, последовательные порты, контроллеры жёстких и гибких дисков.

У центрального процессора есть всего один вход аппаратного прерывания. Поэтому в ПЭВМ все возможные аппаратные прерывания поступают на процессор не напрямую, а через специальное устройство — контроллер прерываний. Это устройство позволяет разрешать и запрещать отдельные прерывания, а также разрешать конфликты между двумя и более одновременно поступившими прерываниями на основе схемы приоритетов.

Центральный процессор, получив сигнал прерывания, «откладывает» текущую выполняемую задачу, сохранив в стеке значения регистров флагов, сегмента кода и счётчика команд, и немедленно переключается на обработку поступившего прерывания. При этом адрес процедуры обработки прерывания хранится в так называемом векторе прерывания, номер которого передаётся процессору вместе с сигналом прерывания.

Вектор прерывания представляет собой четыре байта, содержащих «далёкий» (far) адрес (сегмент и смещение) начала процедуры обработки прерывания. Все векторы прерывания расположены в начале оперативной памяти, начиная с адреса 0000:0000h. Таким образом, для того чтобы вычислить начало любого вектора прерывания в оперативной памяти, достаточно умножить номер вектора прерывания на 4.

Необходимо иметь в виду, что при обработке прерывания процессор будет использовать какие-то из своих регистров. Поскольку

отложенная задача об этом «не догадывается», в начале процедуры обработки прерывания необходимо сохранить используемые регистры в стеке, а перед выходом из прерывания — восстановить их.

Возврат из процедуры обработки прерывания осуществляется командой iret. При этом из стека извлекаются сохранённые при входе в прерывание регистры сегмента кода, счётчика команд и флагов, и процессор возвращается к выполнению прерванной задачи.

Обращаем внимание читателей, что в архитектуре ПЭВМ IBM PC непосредственно перед выходом из процедуры обработки прерывания необходимо выполнить операции по сбросу контроллера прерываний. Иначе повторные прерывания будут невозможны, и система скорее всего не сможет продолжить нормальную работу.

В случае данной лабораторной работы такие действия не потребуются, поэтому интересующихся отправим к книге [2].

1) Общие правила перехвата прерываний

Операционная система MS DOS уже содержит в своём составе обработчики прерываний для всех перечисленных выше аппаратных прерываний. Эти обработчики по выполняемым функциям полностью согласованы с другими частями операционной системы и обеспечивают гарантированно правильное обслуживание соответствующих устройств ПЭВМ.

Однако иногда программисту требуется дополнить стандартную обработку прерывания теми или иными собственными действиями. В этом случае говорят о необходимости перехвата прерывания.

Общая идея этого процесса заключается в том, чтобы запомнить стандартный вектор прерывания где-либо в оперативной памяти, а на его место поместить адрес процедуры, разработанной программистом. Эта процедура может работать по одному из двух алгоритмов.

В первом случае (если команд, добавляемых программистом, немного) процедура должна выполнить свои команды и в конце передать управление стандартной процедуре обработки прерывания. При этом необходимо иметь в виду, что, поскольку стандартная процедура будет осуществлять возврат командой iret, а в стеке уже находятся все три извлекаемые при этом слова, то передача управления должна быть выполнена командой jmp far (безусловный далёкий переход по адресу стандартной процедуры, сохранённому в оперативной памяти).

Во втором случае (как правило, когда добавляется много команд) процедура должна сперва передать управление стандартной процеду-

ре обработки прерывания, дождаться возврата из неё, а уже потом выполнять свои действия. В этом случае по окончании своей работы сама процедура должна осуществить выход из прерывания командой iret. Передача управления стандартной процедуре обработки прерывания здесь представляет определённую хитрость. Дело в том, что в распоряжении программиста есть только один способ вызвать процедуру с возвратом — команда call far. Но эта команда помещает в стек лишь два слова (адрес возврата), а команда, завершающая вызываемую процедуру (iret), предполагает, что в стеке находится три слова (адрес возврата и флаги). Поэтому перед вызовом call far необходимо «вручную» поместить в стек значение регистра флагов командой pushf.

Поскольку в обоих случаях используется стандартная процедура обработки прерывания, все действия по управлению аппаратурой выполняются ею, и программисту нет необходимости об этом задумываться.

Необходимо помнить, однако, что в первом случае, когда программист добавляет команды перед вызовом стандартной процедуры обработки прерывания, добавленные команды выполняются в режиме, когда аппаратура ещё не обслужена. В частности, не сброшен контроллер прерываний, и запрещены все прерывания более низкого приоритета. Именно это обстоятельство требует выполнения минимального количества команд.

Ещё одно важное обстоятельство — нереентерабельность ОС MS DOS. Поскольку аппаратное прерывание может произойти в любой момент времени, в том числе и тогда, когда процессор выполняет код прерываний MS DOS (например, прерывания 21h), в общем случае внутри обработки прерывания нельзя пользоваться сервисом MS DOS без специально принятых мер. Это, в частности, означает невозможность выполнения любых файловых операций.

После того, как написанное программистом «расширение» обработчика прерывания становится ненужным, его необходимо удалить из оперативной памяти, вернув старый вектор обработки прерывания на место.

Замена вектора обработки прерывания и его возврат могут быть осуществлены напрямую командами mov, но в этом случае необходимо на период замены вектора запрещать все прерывания командой cli и по окончанию замены разрешать прерывания командой sti.

Однако лучше использовать для этой цели специальные прерывания MS DOS.

2) Описание используемых прерываний

Для выполнения лабораторных работ используются два аппаратных прерывания:

– 08h — прерывание таймера, имеет наивысший приоритет в системе, «срабатывает» с периодом 55 мс;

– 09h — прерывание клавиатуры, имеет следующий по уровню приоритет, возникает при нажатии и отпускании кнопок на клавиатуре.

По умолчанию в системе эти прерывания всегда разрешены, за исключением случаев запрета всех прерываний командой cli.

Также будет использован вектор прерывания 1Ch. Этот вектор программно вызывается во время обработки прерывания 08h и первоначально указывает на программу обработки, состоящую из единственной команды: iret. Предназначен специально для создания пользовательских дополнений к процедуре обработки прерывания таймера.

Для управления векторами прерываний понадобятся две функции прерывания 21h.

Функция 25h — устанавливает вектор прерывания на указанный адрес. При этом старое значение вектора не сохраняется. В процессе замены вектора все аппаратные прерывания запрещаются.

Вход:

AH=25h

AL=номер заменяемого вектора прерывания

DX=смещение устанавливаемой процедуры обработки прерывания

DS=сегмент устанавливаемой процедуры обработки прерывания

Выход: нет.

Функция 35h — возвращает текущее значение вектора прерывания.

Вход:

AH=35h

AL=номер запрашиваемого вектора

Выход:

BX=значение смещения вектора прерывания

ES=значение сегмента вектора прерывания

2. Задание на лабораторную работу

Написать на ассемблере IBM PC программу перехвата заданного аппаратного прерывания. Программа должна перехватить прерывание и продемонстрировать обработку перехваченного прерывания выводом заданного набора символов на экран прямым доступом к видеопамяти. При этом руководствоваться требованиями к программе, представленными в табл. 2.1.

Разрабатываемая программа не должна становиться резидентной в памяти! После окончания процесса демонстрации программа должна восстановить оригинальное значение вектора прерывания и завершить свою работу с возвратом управления в MS DOS.

3. Содержание отчёта

Отчёт должен содержать:

– титульный лист;

– задание на работу;

– краткое описание алгоритма работы программы, используемых прерываний и функций;

– текст программы.

4. Навыки, полученные студентом

После выполнения лабораторной работы студент должен:

– иметь представление об аппаратных прерываниях IBM PC;

– знать назначение наиболее часто используемых аппаратных прерываний;

– владеть методами перехвата аппаратных прерываний;

– уметь применять аппаратные прерывания для создания собственных программ.

Т а б л и ц а 2.1

Номер в журнале	Номер прерывания	Способ перехвата	Способ отображения
1	08h	A	a
2	09h	B	b
3	1Ch	A	c
4	08h	B	d
5	09h	A	a
6	1Ch	B	b
7	08h	A	c
8	09h	B	d
9	1Ch	A	a
10	08h	B	b
11	09h	A	c
12	1Ch	B	d
13	08h	A	a
14	09h	B	b
15	1Ch	A	c
16	08h	B	d
17	09h	A	a
18	1Ch	B	b
19	08h	A	c
20	09h	B	d
21	1Ch	A	a
22	08h	B	b
23	09h	A	c
24	1Ch	B	d

П р и м е ч а н и е: A — перехват прерывания прямым доступом к вектору прерывания; B — перехват прерывания с помощью функций 25h/35h прерывания 21h; a — инверсия атрибутов символов всего экрана; b — циклическая замена цвета у символов всего экрана; c — печать в центре экрана количества обработанных прерываний; d — вывод строки из 160 символов, начиная с верхнего левого угла экрана, по одному символу за прерывание.

Лабораторная работа 3

СОЗДАНИЕ ПРОСТЕЙШЕГО ПЛАНИРОВЩИКА ПРОЦЕССОВ

1. Общие сведения

Целью лабораторной работы является разработка простейшего планировщика процессов, функционирующего в операционной системе MS DOS и управляющего параллельной работой нескольких простых процессов. При этом планировщик должен работать в реальном режиме процессора 8086 и не должен использовать специальные команды управления процессами защищённого режима, появившиеся в более поздних моделях процессоров этого ряда.

Для целей выполнения лабораторной работы возможны два способа организации многозадачных вычислений:

– выделение каждому процессу кванта времени и принудительное переключение на следующий процесс при окончании кванта;

– «добровольное» освобождение каждым процессом процессора в определённых точках и передача управления следующему процессу.

Важно иметь в виду, что в первом случае возможно как простое циклическое переключение между процессами, так и переключение с учётом приоритетов задач. При этом под приоритетом в данном случае можно понимать количество квантов времени подряд, отводимое процессу для выполнения.

В реальных системах низкоприоритетные процессы получают возможность выполняться, лишь когда все высокоприоритетные процессы заблокированы. Но в нашем случае процессы не блокируются. Впрочем, в нашем случае можно реализовать схему с накоплением приоритета, когда при каждом переключении всем процессам, не выбранным для выполнения, приоритет повышается на единицу, на выполнение выбирается процесс с наибольшим накопленным приоритетом, а отработавший процесс получает приоритет, заданный ему при запуске.

Во втором случае процесс, освобождающий процессор, может как просто передать управление следующему процессу (произвольному, оказавшемуся первым в очереди), так и указать явно, какому процессу он передаёт управление.

В обоих случаях вместо очереди процессов может быть также реализован механизм случайной выборки процесса из неупорядоченного множества готовых к выполнению процессов.

В любом случае выполняющиеся процессы не могут просто передавать управление друг другу с помощью команд безусловного перехода, так как в общем случае в произвольно заданный момент времени передачи управления неизвестно, в какую точку другой программы должно быть передано управление. Поэтому для переключения процессов и требуется планировщик, отслеживающий состояние каждого процесса в момент переключения. Это состояние, включающее в себя содержимое регистров процессора и сопроцессора, состояние стека и различную другую информацию, обычно называют контекстом процесса.

Важно понимать, что одним из важнейших элементов контекста является именно состояние стека, так как процесс может в процессе своей работы вызывать подпрограммы, обращаться к системным подпрограммам обслуживания и т. п. И поскольку переключение на другой процесс может произойти в любой момент, сохранить текущее состояние стека очень важно для возобновления данного процесса в дальнейшем. Иными словами, мы приходим к необходимости создания отдельного стека для каждого запускаемого процесса.

При переключении с одного процесса на другой планировщик должен сохранить контекст текущего процесса. Очевидно, что в простейшем случае наилучшим местом для сохранения контекста является сам стек текущего процесса. Далее планировщик должен выбрать из очереди следующий процесс, сделать его стек текущим, восстановить из него контекст этого процесса и передать ему управление.

1) Планировщик с принудительным переключением

В случае планирования процессов с принудительным переключением наилучшим условием для переключения процессов является прерывание таймера (08h). В этом случае квант времени, выделяемый каждому процессу, равен 55 мс, что для большинства приложений вполне удовлетворительно.

Планировщик базируется на идее перехвата прерывания 08h. После входа в прерывание стек текущего процесса содержит три слова — флаги, сегмент кода и счётчик команд. Если сейчас подать команду iret, то процесс возобновит свою работу.

Перед переключением на другой процесс планировщик должен сохранить в стеке значения всех регистров процессора, кроме SS:SP, которые как раз и указывают на текущее состояние стека процесса. Значения этих регистров должны быть сохранены в оперативной памяти, выделенной планировщиком для хранения информации о запущенном процессе. Далее в регисты SS:SP планировщик должен занести соответствующие значения, описывающие стек возобновляемого процесса, и восстановить из него значения всех остальных регистров (естественно, кроме счётчика команд, сегмента кода и флагов). После чего новый процесс готов продолжить выполнение по команде iret.

Таким образом, главной проблемой является начальное формирование стека запускаемого процесса. Такое, чтобы он мог быть запущен в работу стандартной процедурой переключения с процесса на процесс, описанной в предыдущем абзаце. Естественно, кроме стека, для каждого процесса должны быть созданы дополнительные переменные, позволяющие планировщику узнать, что данный процесс нужно учитывать при планировании, а также информацию о его приоритете и т. п.

2) Планировщик с «добровольным» переключением

В этом случае каждый процесс должен быть разработан с учётом того, что время от времени он сам должен передавать управление другим процессам. Но на практике он может это делать, лишь передавая управление планировщику. Обычно это реализуется посредством специального системного вызова.

В этом случае наиболее удобно планировщик реализовать также как процедуру обработки прерывания. Тогда будут справедливы все рассуждения, сделанные нами в предыдущем разделе. А в качестве системного вызова переключения на другой процесс может выступать программное прерывание.

В операционной системе MS DOS выделена специальная группа программных прерываний, предназначенная для пользовательских процедур. В лабораторной работе рекомендуется использовать для этой цели прерывание 60h.

3) Важное замечание

Описанным способом можно создать параллельно выполняющиеся процессы любой степени сложности. Однако на практике необходимо принимать специальные меры для обхода нереентерабельности

системных вызовов MS DOS. В силу сложности этой процедуры в данной лабораторной работе предполагается выполнение лишь простейших параллельных процессов, не использующих сервиса системных вызовов.

2. Задание на лабораторную работу

Написать на ассемблере IBM PC программу простейшего планировщика, обеспечивающего параллельное выполнение нескольких процессов.

В качестве процессов использовать циклически работающие процедуры, включённые в текст программы планировщика.

Для демонстрации своей работы процессы должны выполнять какие-либо преобразования в видеопамяти с использованием прямого доступа к ней (не используя сервис прерываний). Конкретный вид преобразований студенты должны подобрать самостоятельно. Рекомендуется использовать требования к программе табл. 2.1, адаптируя их к количеству процессов.

При выполнении работы руководствоваться требованиями к программе, представленными в табл. 3.1.

3. Содержание отчёта

Отчёт должен содержать:

– титульный лист;

– задание на работу;

– краткое описание алгоритма работы программы, используемых прерываний и функций;

– текст программы.

4. Навыки, полученные студентом

После выполнения лабораторной работы студент должен:

– иметь представление об основах организации многозадачной работы ЭВМ;

– знать основные стратегии планирования процессов;

– владеть методами практического создания многозадачных систем;

– уметь организовывать процесс отладки и тестирования в программах с параллельно выполняющимися процессами.

Т а б л и ц а 3.1

Номер в журнале	Способ вызова планировщика	Способ выборки процесса	Количество процессов
1	08h	б	2
2	60h	б	3
3	08h	ж	4
4	60h	ж	2
5	08h	н	3
6	60h	н	4
7	08h	с	2
8	60h	с	3
9	08h	б	4
10	60h	б	2
11	08h	ж	3
12	60h	ж	4
13	08h	н	2
14	60h	н	3
15	08h	с	4
16	60h	с	2
17	08h	б	3
18	60h	б	4
19	08h	ж	2
20	60h	ж	3
21	08h	н	4
22	60h	н	2
23	08h	с	3
24	60h	с	4

П р и м е ч а н и е: 08h — принудительное переключение; 60h — «добровольное» переключение; б — циклическая очередь без приоритетов; ж — очередь с жёстко заданными приоритетами; н — очередь с накапливаемыми приоритетами; с — случайная выборка.

Библиографический список

1. Григорьев В. Л. Видеосистемы ПК фирмы IBM. — М.: Радио и связь, 1993. — 192 с.
2. Джордейн Р. Справочник программиста персональных компьютеров типа IBM PC, XT и AT: Пер. с. англ. Н. В. Гайского. — М.: Финансы и статистика, 1992. — 544 с.

Содержание

Предисловие 3
Лабораторная работа 1. Сравнение реальной и расширенной машин 4
Лабораторная работа 2. Обработка аппаратного прерывания 12
Лабораторная работа 3. Создание простейшего планировщика процессов 18
Библиографический список 23