Лабораторная работа №2

Исследование общих принципов реализации многозадачности

Работа проводится с использованием программной модели планировщика, реализующей:

- поддержку (псевдо)параллельного выполнения до 10 потоков программы;
- выполнение потоков в режиме вытесняющий и невытесняющей многозадачности;
- поддержку механизма критических секций (до 10 критических секций).

Модель планировщика реализована на языке Турбо Паскаль 7.0 для DOS и представляет собой модуль (unit) Турбо Паскаля, написанный с использованием вставок на языке ассемблера процессора Intel 8086/88.

> **Внимание!** На современных машинах (более мощных, чем PII-400MHz) при использовании в Паскаль-програме модуля CRT возникает ошибка 200 "Деление на ноль". Для устранения этой неприятности необходимо пользоваться исправленным модулем CRT или прилагаемой "заплаткой" **rdelay.tpu**. Данный модуль должен фигурировать во фразе **uses** основной программы непосредственно перед **crt**; кроме того, компиляция обязательно должна осуществляться на диск.

Теоретические положения

Термины и определения

Планировщиком называется часть операционной системы, ответственная за организацию (псевдо)параллельного выполнения нескольких задач и потоков в рамках одной задачи. Изучаемая модель планировщика управляет потоками.

Задачей (*task, application*) или **процессом** (*process*) называется выполняемая программа. В многозадачной ОС могут одновременно выполняться несколько задач в режиме разделения процессорного времени (или действительно одновременно, если процессоров несколько). Каждая задача имеет свой стек и, как правило, собственное адресное пространство; для задач планировщик в специальной записи состояния отслеживает выделенные ресурсы (открытые файлы, занятые блоки памяти и т.д.)

Потоками (*threads*) называются ветви (процедуры) в рамках одной задачи, выполняемые параллельно. Потоки разделяют ресурсы задачи, выполняются в одном адресном пространстве, но каждый поток имеет свой собственный стек.

Термин "**невытесняющая** (*non-preemptive*) **многозадачность**" означает, что переключение процессора с одной задачи на другую происходит, когда выполняемая в данный момент задача явно или неявно (в любом случае – через вызов каких-то системных функций) передает управление планировщику, чтобы тот мог выполнить переключение.

При **вытесняющей** (*preemptive*) **многозадачности** запуск процедуры переключения происходит независимо от "желания" текущей задачи, текущая задача принудительно прерывается в произвольный момент, и активизируется другая. Для инициации переключения задач обычно используется аппаратное прерывание от таймера.

Критической секцией (КС) называется участок кода некоторой программы А, выполняемой обычно в режиме вытесняющей многозадачности, в пределах которого запрещено выполнение другими параллельно исполняемыми программами некоторых действий, которые могут помешать правильной работе рассматриваемого участка кода программы А. В виде критических секций обычно оформляются последовательности команд, для которых

важно соблюсти неразрывность выполнения "одна-за-одной". При этом **вход в критическую секцию НЕ означает, что задача не может быть прервана и управление не может быть передано другой задаче!**

Критическая секция начинается командой входа в критическую секцию и заканчивается командой выхода. Планировщик анализирует запрос процесса на вход в критическую секцию и, если вход возможен, начинают выполняться команды критической секции. Если вход в критическую секцию невозможен, т.е. какой-либо другой процесс находится в состоянии выполнения критической секции, запрашивающий процесс переводится в состояние ожидания и выполняются другие процессы, пока не сложится ситуация, делающая для запрашивающего процесса возможным вход в критическую секцию.

Многие операционные системы, например - Windows 95, поддерживают нумерованные критические секции. С их помощью можно осуществить разделение доступа процессов к определенным ресурсам, не блокируя работу тех процессов, которые не используют ресурс. Для этого процессы должны при входе в КС указывать идентификатор (номер) КС. Если некоторый процесс выполняет КС с определенным номером, то другие процессы не могут войти в КС с тем же номером, но могут выполнять КС с другими номерами. Например, пусть для вывода на печать процесс должен войти в КС1, а для вывода в файл - в КС2. При этом, если один процесс печатает (выполняет команды, защищенные КС1), то ничто не мешает другому процессу работать с файлом (выполнять КС2), но начать печатать (войти в КС1) он сможет только тогда, когда первый процесс выйдет из КС1.

Команды входа и выхода из критической секции представляют собой так называемые "**критические скобки**".

Использование модели планировщика для организации параллельных вычислений

Изучаемая в данной работе модель планировщика реализует лишь часть функций планировщика реальной ОС. Модель написана для процессора Intel 8086/88 (или, соответственно, для реального режима работы более старших процессоров данного семейства). В связи с этим функции выделения памяти и поддержка выполнения нескольких задач не реализованы, реализовано лишь выполнение процедур DOS-программы в режиме разделения времени с использованием механизма невытесняющей или вытесняющей многозадачности и критических секций. В режиме вытесняющей многозадачности планировщик циклически выделяет всем активным процессам одинаковые кванты времени.

Планировщик реализован в качестве множества функций модуля H314TASK.TPU (исходный текст - файл H314TASK.PAS). Поэтому в разделе **uses** основной программы должен упоминаться этот модуль.

Каждый параллельный поток основной программы реализуется в виде процедуры без параметров, обязательно откомпилированной с дальней моделью вызова (опция компилятора $F+). Обычно тело процедуры представляет собой цикл, конечный или бесконечный. Рекомендуется отключить проверку переполнения стека (опция компилятора $S-), так как планировщик переключает стек в процессе своей работы. Процедуры ни в коем случае не должны завершаться, поэтому рекомендуется в конце каждой процедуры-потока ставить вызов процедуры LoopBack, зацикливающей поток (возврат из процедуры LoopBack никогда не производится).

Для запуска параллельных вычислений необходимо выполнить следующие действия:

1) Зарегистрировать потоки в таблице потоков планировщика при помощи функции

```
function RegisterThread(ThreadAddr:pointer; StackSize:word):integer;
```

Параметры функции - адрес точки входа в процедуру потока и размер стека потока. Функция возвращает хэндл (номер) потока, который идентифицирует его в системе. Например, можно написать:

```
ht1 := RegisterThread(@proc1,8000);
```

2) Запустить выполнение зарегистрированных потоков в режиме разделения времени, выполнив процедуру

```
procedure ExecuteRegisteredThreads(UseTimer:boolean);
```

Параметр UseTimer определяет, будет ли планировщик производить переключение задач по прерываниям таймера (TRUE, вытесняющая многозадачность) или для этого в теле каждой задачи должна периодически вызываться процедура SwitchThreads (FALSE, невытесняющая многозадачность).

3) Очистить информацию в таблице потоков планировщика и освободить память, занимаемую потоками под стек, с помощью процедуры ClearThreads.

При запуске процедуры ExecuteRegisteredThreads выполнение дальнейших команд основного потока программы приостанавливается до тех пор, пока все параллельные потоки не будут завершены вызовом процедуры StopThread с параметром – номером останавливаемого потока (возвращается процедурой RegisterThread). Процедура StopThread может вызываться из любого параллельного потока, а не только из того, который останавливается. Это позволяет логически выделить один из параллельных потоков в качестве "главного" (управляющего) и сосредоточить управление параллельными вычислениями в нем.

Для управления выполнением потоков предусмотрены следующие процедуры:

StopThread(num: word); - останавливает выполнение потока, запрещая передачу ему управления планировщиком. Передача управления может быть восстановлена процедурой RunThread. Когда все потоки остановлены, выполнение процедуры ExecuteRegisteredThreads завершается и продолжается выполнение вызвавшего ее основного потока программы.

RunThread(num:word); - возобновляет выполнение потока, ранее остановленного командой StopThread, включая его в число потоков, обслуживаемых планировщиком. Управление запускаемому потоку не передается немедленно, он активизируется "на общих основаниях" в соответствии с алгоритмом работы планировщика.

SwitchThreads; - передает управление планировщику для активизации им другого потока. Эта процедура в режиме невытесняющей многозадачности всегда должна периодически вызываться в теле потоков для обеспечения переключения задач. В режиме вытесняющей многозадачности вызов этой процедуры позволяет "досрочно" отдать управление другому потоку, например, если текущий поток в цикле ожидает какого-либо события, в противном случае ожидающий поток будет просто впустую расходовать время процессора.

LoopBack; - бесконечный цикл, из которого нет выхода. Эту процедуру рекомендуется вызывать в конце каждой процедуры потока.

EnterCritical(num: integer); - вход процесса в критическую секцию с указанным номером. Если вход возможен, поток продолжает выполняться дальше, блокируя вход в критическую секцию с данным номером для всех остальных потоков. Если вход невозможен, то управление передается другим потокам, и выполнение команд потока возобновится только тогда, когда блокирующий критическую секцию поток освободит ее.

LeaveCritical(num: integer); - выход процесса из критической секции с указанным номером. После выполнения этой команды другие процессы получают возможность войти в критическую секцию с указанным номером. Если на момент выхода из КС в системе имеются процессы, ожидающие входа в эту КС, то покидающий КС процесс досрочно прерывается и активизируется следующий процесс (не обязательно ожидающий входа в КС, просто - следующий).

Пример программы, использующей описываемый планировщик, можно найти в файле TASKSWAP.PAS.

Реализация планировщика

Поскольку наш планировщик не занимается выделением потокам ресурсов (за исключением стека), то основной его функцией является обеспечение переключения между выполняемыми параллельно потоками.

Состояние потока в каждый момент времени описывается состоянием регистров процессора. Прервав выполнение команд одного потока с помощью аппаратного или программного прерывания, можно затем организовать подмену значений в регистрах на значения для другого потока, после чего выход из процедуры обслуживания прерывания будет производиться уже в новый участок кода, соответствующий другому потоку.

Для поддержания информации о потоках планировщик использует 29-байтные записи (массив TS, записи типа TThreadStateWord), содержащие значения всех регистров процессора (в том числе флагов, сегментных регистров, счетчика команд и указателя стека), а также признак активности потока, указывающий, следует ли планировщику выделять время этому потоку. Информация о размерах стеков потоков содержится в массиве Stacks.

Процедура переключения потоков назначается в качестве обработчика прерывания 8, возникающего аппаратно по таймеру (процедура TimerHandler). Эта процедура извлекает состояние регистров, бывшее до вызова прерывания, из стека и сохраняет их в записи таблицы потоков (TS), соответствующей активному потоку. Затем определяется новый поток из числа активных, который подлежит активизации (перебор записей в таблице потоков производится циклически, что гарантирует, что каждый активный поток рано или поздно получит управление). Из записи таблицы потоков, соответствующей активизируемому потоку, извлекается состояние регистров (значения большинства регистров подменяются в стеке, так как будут извлекаться оттуда при возврате из процедуры прерывания, значение в паре SS:SP, адресующей стек, заменяется непосредственно). Возврат из обработчика прерывания производится уже в новый поток с восстановленными значениями его регистров.

> Некоторые сведения о работе компилятора:
>
> Наш обработчик скомпилирован с ключевым словом **interrupt**. Это означает, что точка входа и точка выхода будут оформлены компилятором следующим образом:
>
> Вход:
>
> (Перед этим при прерывании в стек были помещены точка возврата (значения в CS:IP) и регистр флагов, прерывания также автоматически запрещены.)

```
push   ax
push   bx
push   cx
push   dx
push   si
push   di
push   ds
push   es
push   bp
```

```
mov    bp,sp     ; сохранение SP в BP
mov    ax,@data  ; настройка DS на сегмент данных программы
mov    ds,ax
.............
```

Выход:

```
pop    bp
pop    es
pop    ds
pop    di
pop    si
pop    dx
pop    cx
pop    bx
pop    ax
iret  ; извлечение флагов и CS:IP
```

Для реализации "досрочного" переключения потоков и невытесняющей многозадачности реализована процедура SwitchThreads, которая просто генерирует прерывание 8 программным путем, при этом взводится флаг DisableHardwareEvents, блокирующий вызов системного обработчика прерывания 8, в котором производится сброс контроллера прерываний и обновление текущего времени: ведь прерывание вызвано не аппаратурой, а программой. Функции по переключению потоков при этом выполняются. Другой флаг, PreemptiveSwitch, позволяет блокировать переключение потоков по аппаратному прерыванию таймера, что дает возможность реализовать невытесняющую многозадачность.

Для поддержания информации о состоянии потоков используется массив записей, имеющих следующую структуру:

```
type TThreadStateWord=record
       BP_reg, ES_reg, DS_reg, DI_reg, SI_reg, DX_reg,
       CX_reg, BX_reg, AX_reg, IP_reg, CS_reg, Flags:  word;
       Stack:pointer;
       Active:boolean;
     end;
```

Порядок следования регистров в записи в точности повторяет порядок их следования в стеке при вызове процедуры обработки прерывания, что позволяет удобно манипулировать регистрами, Stack содержит текущее состояние пары SS:SP, Active показывает, активен процесс или остановлен.

При регистрации потока процедурой RegisterThread значения регистров в соответствующей записи обнуляется, в поле Active заносится TRUE, под стек потока в куче захватывается блок памяти указанного размера. В поле Stack заносится указатель на последнее слово в указанном блоке, так как при нарастании стека указатель стека SP уменьшается. Значение поля Stack будет выбираться в качестве начального состояния стека (SS:SP) соответствующего потока, в нем же будет сохраняться состояние стека потока при переключении потоков.

Действия процедуры ExecuteRegisteredThread состоят в том, что производится очистка таблиц критических секций, инициализируется обработчик прерывания 8, после чего запускается цикл ожидания. Условием выхода из цикла ожидания является значение TRUE в переменной ParallelFinished, свидетельствующее о том, что все зарегистрированные потоки остановлены. При выходе восстанавливается старое значение обработчика прерывания 8. Если происходит запуск параллельных вычислений в режиме невытесняющей многозадачности, то перед входом в цикл управление передается одному из потоков командой SwitchThreads, так как по аппаратным прерываниям таймера переключение в этом случае не производится.

Критические секции

Для реализации механизма критических секций служат процедуры EnterCritical и LeaveCritical. Разумеется, использование критических секций имеет смысл только при вытесняющей многозадачности, в случае невытесняющей многозадачности просто не следует отдавать управление планировщику в критических участках кода.

> В данном случае, когда программа написана для реального режима DOS, можно защитить критический участок кода от какого бы то ни было вмешательства, просто запретив прерывания командой CLI. Это так, но в реальных многозадачных операционных системах пользовательская программа как правило не имеет доступа к системе прерываний вообще, а команды типа CLI/STI вызывают исключительную ситуацию в процессоре, обрабатываются ОС и в конце концов к общему запрещению прерываний как правило не приводят, в лучшем случае – сигналы о прерываниях перестают поступать в приложение.

Внутри планировщика поддерживаются два вспомогательных массива: CriticalTable и CriticalTableWait. Индекс в массиве соответствует состоянию критической секции с соответствующим номером. Если ячейка в массиве CriticalTable содержит 0, то любой процесс способен сразу войти в критическую секцию с соответствующим номером выполнив процедуру EnterCritical. При этом значение в ячейке будет изменено на ненулевое.

Если значение в ячейке CriticalTable при выполнении EnterCritical не равно 0, то процесс блокируется перед входом в критическую секцию и ожидает ее освобождения, а для индикации наличия ожидающих процессов увеличивается счетчик в соответствующей массива CriticalTableWait. Так как процесс переходит в состояние ожидания, выполняется процедура SwitchThreads, активизирующая следующий процесс, независимо от того, разрешены ли аппаратные прерывания от таймера.

При выполнении процедуры LeaveCritical соответствующая в массиве CriticalTable сбрасывается в 0. Если при этом в соответствующей ячейке массива CriticalTableWait содержится ненулевое значение, свидетельствующее о наличии ожидающих процессов, то происходит передача управления следующему процессу с помощью SwitchThreads. В противном случае, если ни один процесс не ожидает входа в освобождаемую критическую секцию, текущий процесс продолжает свое выполнение.

Ситуация взаимной блокировки

При использовании механизмов блокировки процессов, к которым относятся критические секции, возможна ситуация, когда два или более процесса блокируют друг друга, т.е. не могут далее выполняться, так как для продолжения одного нужно, чтобы другой освободил некий ресурс, а другой процесс ожидает аналогичного действия от первого.

Простейшим примером такой потенциально опасной ситуации может служить следующая ситуация:

Процесс 1	**Процесс 2**
EnterCritical(1);	EnterCritical(2);
EnterCritical(2);	EnterCritical(1);
..... действия секции	..... действия секции .....
LeaveCritical(1);	LeaveCritical(1);
LeaveCritical(2);	LeaveCritical(2);

В этом случае возможен, например, вариант, когда после входа в критическую секцию 1 первый процесс прерывается, начинает исполняться второй процесс, там происходит вход в КС2 и остановка в ожидании освобождения КС1, занятой первым процессом. Когда же первый процесс получит управление, он продолжит свое выполнение и попытается войти в КС2, только что занятую вторым процессом, и также остановится в ожидании. В результате выполнение процессов 1 и 2 будет остановлено.

Очевидно, что ситуация может развиваться и иначе: если между входом в 1 и 2 КС первого процесса не произойдет прерывания, то блокировки не будет.

В изложенной ситуации ясен и путь решения проблемы: если захватывать и освобождать КС в обоих процессах в одинаковой последовательности, то взаимная блокировка будет невозможна.

К сожалению, в реальной жизни встречаются более запутанные ситуации со взаимной блокировкой нескольких процессов, и разрешить ее бывает не так просто, особенно если учесть, что условие для наступления блокировки наступает случайно и может выполняться достаточно редко. Это приводит к трудности в обнаружении причины остановки программы - все выглядит так, как будто программа работает, а потом, изредка, без всяких видимых причин зависает.

Задание

Выполняется при самостоятельной подготовке:

Изучить теоретическую часть описания ЛР и материал соответствующих лекций.

Выполняется в лаборатории:

1. Включить компьютер. Если на компьютере установлена операционная система Windows 95/98 - запустить ее в режиме эмуляции DOS. Запустить Турбо Паскаль 7.0.

2. Загрузить исходный текст примера TASKSWAP.PAS, изучить логику использования предлагаемой программной модели планировщика.

3. Загрузить исходный текст модуля планировщика H314TASK.PAS, изучить логику реализации планировщика.

4. Смоделировать ситуацию взаимной блокировки, написав для этого программу, использующую модуль H314TASK. Отладить и запустить программу, продемонстрировать результаты работы преподавателю.

5. Написать и отладить программу по индивидуальному заданию (см. ниже). Продемонстрировать результаты работы преподавателю.

6. Завершить работу в Турбо Паскале. Выключить компьютер.

Варианты индивидуальных заданий

Написать программу, выполняющую в режиме разделения времени указанные в таблице три процесса. Расшифровка потоков приводится ниже:

Вариант	Многопоточность	Поток 1	Поток 2	Поток 3
1	вытесняющая	A	C	E

2	вытесняющая	B	C	F
3	вытесняющая	A	D	F
4	вытесняющая	B	D	E
5	вытесняющая	A	B	D
6	невытесняющая	B	C	E
7	невытесняющая	A	C	F
8	невытесняющая	B	D	F
9	невытесняющая	A	D	E
10	невытесняющая	B	A	C

A. Поток ожидает нажатия клавиши Esc (#27), при ее нажатии должны завершаться все потоки. См. функции KeyPressed и ReadKey.

B. Поток завершается спустя количество секунд, равное номеру варианта, умноженному на 5. При его завершении должны завершаться все потоки.

Примечание: Использовать функцию Delay Паскаля для организации задержки нельзя, так как она блокирует прерывания.

C. Поток рисует на экране линии со случайными координатами обоих концов и цветом.

D. Поток рисует на экране линии от конца предыдущей нарисованной линии до точки со случайными координатами и цветом.

E. Поток рисует на экране 1000 точек со случайными координатами одного, общего для этой итерации, случайного цвета, потом рисует на их месте черные точки (стирает их), потом цикл повторяется.

F. Поток рисует на экране эллипсы случайного цвета, размера и положения.