Лабораторная работа № 4

СОЗДАНИЕ ПРОЦЕССОВ И ПОТОКОВ

4.1.Цель работы

Целью работы является изучение основных принципов организации многозадачных операционных систем. Все многозадачные операционные системы используют концепцию процесса и потока. В данной работе рассматриваются следующие вопросы:

* Чередование выполнения нескольких процессов с целью повышения степени использования процессора;
* Разделение ресурсов между процессами;
* Организация обмена данными между процессами и потоками;
* Изменение класса приоритета процесса и уровня приоритета потока.

4.2. Теоретическое введение

В теоретическом введении рассматриваются процедуры и функции ядра ОС Windows, применяемые для работы с процессами и потоками.

4.2.1. Создание процесса

Для создания процесса используется функция CreateProcess, имеющая следующий вид:

function CreateProcess (lpApplicationName: PChar; lpCommandLine: PChar;

lpProcessAttributes, lpThreadAttributes: PSecurityAttributes;

bInheritHandles: BOOL; dwCreationFlags: DWORD;

lpEnvironment: Pointer; lpCurrentDirectory: PChar;

const lpStartupInfo: TStartupInfo;

var lpProcessInformation: TProcessInformation): BOOL; stdcall;

Входные параметры данной функции перечислены в таблице 4.1.

Таблица 4.1.

Параметры функции CreateProcess

|  |  |
| --- | --- |
| Параметр | Описание |
| lpApplicationName | Имя исполняемого файла |
| lpCommandLine | Командная строка, передаваемая создаваемому процессу. |
| lpProcessAttributes | Определяют наследование дочерним процессом свойств родительского процесса. Обычно lpProcessAttributes и lpThreadAttributes устанавливают равным nil, а bInheritHandles =false. |
| lpThreadAttributes |
| bInheritHandles |
| dwCreationFlags | Флаги, определяющие характеристики данного процесса. Например, его приоритет. |
| lpEnvironment | Указывает на блок памяти, хранящий строки переменных окружения. Обычно устанавливают в nil. |
| lpCurrentDirectory | Строка, определяющая текущий каталог и диск. Если этот параметр - nil, рабочий каталог нового процесса будет тем же, что и у приложения породившего его. |
| lpStartupInfo | Указатель на структуру TStartupInfo. Из всех полей данной структуры обязательным является только заполнение поля cb- размер в байтах данной структуры. |
| lpProcessInformation | Указатель на структуру TProcessInformation, из которой родительский процесс может получить данные о создаваемом процессе. |

Структура TStartupInfo имеет следующий вид:

TStartupInfo = record

cb: DWORD; lpReserved: Pointer; lpDesktop: Pointer;

lpTitle: Pointer; dwX: DWORD; dwY: DWORD;

dwXSize: DWORD; dwYSize: DWORD;

dwXCountChars: DWORD; dwYCountChars: DWORD;

dwFillAttribute: DWORD; dwFlags: DWORD;

wShowWindow: Word; cbReserved2: Word; lpReserved2: Byte;

hStdInput: THandle; hStdOutput: THandle; hStdError: THandle;

end;

Описание полей структуры TStartupInfo приведены в таблице 4.2.

Таблица 4.2.

Основные поля структуры TStartupInfo

|  |  |
| --- | --- |
| Поле | Описание |
| cb | Размер структуры в байтах. Служит для контроля версий. |
| lpReserved | Зарезервировано |
| lpDesktop | Определяет имя рабочего стола, на котором запускается приложение. |
| lpTitle | Используется для консольных приложений и определяет заголовок консольного окна. Если этот параметр – nil, то в заголовок выводится имя исполняемого файла. |
| dwX , dwY | X иY координаты левого верхнего угла окна приложения. Измеряется в пикселях |
| dwXSize, dwYSize | Ширина и высота (в пикселях) окна приложения |
| dwXCountChars,  dwYCountChars | Определяют ширину и высоту (в символах) консольных окон |
| dwFillAttribute | Задает цвет текста и фона в консольных приложениях |
| dwFlags | Содержит набор флагов, позволяющих управлять созданием дочернего процесса |
| wShowWindow | Определяет вид окна приложения |
| cbReserved2 | Зарезервировано |
| lpReserved2 | Зарезервировано |
| hStdInput,  hStdOutput  hStdError | Описатели буферов для консольного ввода/вывода |

Структура TProcessInformation имеет следующий вид:

TProcessInformation =record

hProcess: THandle; hThread: THandle;

dwProcessId: DWORD; dwThreadId: DWORD;

end;

Описание полей структуры TProcessInformation приведены в таблице 4.4. Структура TProcessInformation содержит выходные данные и заполняется функцией CreateProcess.

Таблица 4.4.

Основные поля структуры TProcessInformation

|  |  |
| --- | --- |
| Поле | Описание |
| hProcess | Дескриптор созданного процесса |
| hThread | Дескриптор первичного потока |
| dwProcessId | Идентификатор процесса |
| dwThreadId | Идентификатор первичного потока |

Если функция CreateProcess успешно выполнена, она возвращает ненулевое значение (true). Если произошла ошибка – возвращается 0 (false). Тогда информацию об ошибке можно получить, вызвав функцию GetLastError.

4.2.2. Запуск внешней программы функциейWinExec

Самый простой способ запустить какую-то программу из своего приложе­ния — использовать функцию **WinExec.**Эта функция может работать в любых версиях Windows и выполнять любые файлы: приложения Windows, MS-DOS, файлы PIF и т.п. Функция **WinExec**определяется следующим образом:

function WinExec(CmdLine: PChar; CmdShow: integer): integer;

Параметр CmdLine является указателем на строку с нулевым символом в кон­це, содержащую имя выполняемого файла и, если необходимо, параметры коман­дной строки. Если имя файла указано без пути, то Windows будет искать в катало­гах выполняемый файл в следующей последовательности:

1. Каталог, из которого загружено приложение
2. Текущий каталог
3. Системный каталог Windows, возвращаемый функцией GetSystemDirectory.
4. Каталог Windows, возвращаемый функцией GetWindowsDirectory
5. Список каталогов из переменной окружения PATH

Параметр CmdShow определяет форму представления окна запускаемого при­ложения Windows. Чаще всего используется значение SW\_RESTORE**,**при котором окно за­пускаемого приложения активизируется и отображается на экране.

При успешном выполнении запуска приложения функция WinExec возвраща­ет значение, большее 31. При неудаче могут возвращаться значения, приведенные в таблице 4.4.

Таблица 4.4.

Коды ошибок завершения функции WinExec

|  |  |  |
| --- | --- | --- |
| Значение | Номер | Описание |
| 0 | 0 | He хватает памяти или ресурсов системы |
| ERROR\_BAD\_FORMAT | 11 | Ошибочный exe- файл (например, не для Win32.EXE) |
| ERROR\_FILE\_NOT\_ FOUND | 2 | Указанный файл не найден |
| ERROR\_PATH\_NOT\_ FOUND | 3 | Указанный каталог не найден |

Достоинством функции WinExec является ее совместимость с ранними версиями Windows. Для этого она и сохраняется в Win32, хотя для Win32 рекомендуется пользоваться функцией CreateProcess.

**4.2.4. Запуск внешней программы и открытие документа функцией ShellExecute**

Функция ShellExecute может не только выполнять заданное приложение, но и открывать документ и печатать его. Под термином «открыть файл документа» понимается выполнение связанного с ним приложения и загрузка в него этого до­кумента. Например, обычно с документами, имеющими расширение .doc, связан Word. В этом случае открыть файл, например, с именем file.doc означает запус­тить Word и передать ему в качестве параметра имя файла file.doc. Кроме описан­ных возможностей функция ShellExecute позволяет открыть указанную папку. Это означает, что будет запущена программа «Проводник» с открытой указанной папкой.

Для использования функции ShellExecute в операторе uses надо добавить мо­дуль ShellAPI. Автоматически Delphi не включает этот модуль в программу.

Функция ShellExecute определена следующим образом:

function ShellExecute(Wnd: HWnd; Operation, FileName, Parameters,

Directory: PChar; ShowCmd: Integer): THandle;

Параметр Wnd является дескриптором родительского окна, в котором отобра­жаются сообщения запускаемого приложения. Обычно в качестве него можно про­сто указать Handle.

Параметр Operation указывает на строку с нулевым символом в конце, кото­рая определяет выполняемую операцию. Эта строка может содержать текст «open» (открыть) или «print» (напечатать). Для Windows 95/98 и NT определено еще одно значение: «explore» (исследовать) — открыть папку. Если параметр Operation ра­вен nil, то по умолчанию выполняется операция «open».

Параметр FileName указывает на строку с нулевым символом в конце, кото­рая определяет имя открываемого файла или имя открываемой папки.

Параметр Parameters указывает на строку *с*нулевым символом в конце, кото­рая определяет передаваемые в приложение параметры, если FileName определяет выполняемый файл. Если FileName указывает на строку, определяющую откры­ваемый документ или папку, то этот параметр задается равным nil.

Параметр Directory указывает на строку с нулевым символом в конце, кото­рая определяет каталог по умолчанию.

Параметр ShowCmd определяет режим открытия указанного файла. Этот па­раметр может иметь множество различных значений. Обычно, как и для функции WinExec, используется значение SW\_RESTORE, при котором окно запускаемого приложения активизируется и отображается на экране.

При успешном выполнении функция ShellExecute возвращает целое значе­ние, большее 32. Значение меньшее или равное 32 указывает на ошибку. Значения эти те же, что и для функции WinExec.

Примеры использования функции ShellExecute.

1. Следующий опера­тор открывает файл документа с именем file.doc, т.е. запускает WinWord (обычно он связан с файлами .doc), загрузив в него указанный файл:

ShellExecute (Handle, nil, ‘file.doc’, nil, nil, SW\_RESTORE) ;

1. Для вывода на печать документа, записывается аналогич­ный оператор, но изменяется значение параметра Operation:

ShellExecute (Handle, ‘print’*,’*file .doc ‘, nil, nil, SW\_RESTORE) ;

Выполнение этого оператора будет протекать следующим образом. Запустится WinWord, связанный с файлами .doc, в него загрузится файл file.doc, затем из Winword запустится печать с атрибутами по умолчанию, после чего файл file.doc выгрузится из Winword.

1. Оператор открывает приложение Windows «Калькулятор»:

ShellExecute (Handle, ‘open’ , ‘Calc’ , nil, nil, SW\_RESTORE) ;

1. Следующий пример открывает папку c:\Program Files\Borland:

ShellExecute (Handle, ‘explore’, ‘c:\Program Files\Borland', nil, nil,SW\_RESTORE) ;

Функция ShellExecute автоматически отыскивает приложение, связанное с типом открываемого документа, и запускает его. Но иногда требуется вызвать явным образом приложение, связанное с каким-то документом, например, чтобы передать ему какие-то дополнительные параметры. Помочь в этом может функция FindExecutable, которая возвращает имя и путь приложе­ния, связанного с указанным файлом. Использование этой функции требует вклю­чения в оператор uses модуля ShellAPI.

Функция FindExecutable определена следующим образом:

function FindExecutable (FileName, Directory, Buffer: PChar): THandle;

Она позволяет получить имя выполняемого exe- файла, связанного с файлом, указанным параметром FileName. Параметр Directory определяет каталог по умолчанию. Оба параметра являются указателями на строки с нулевым символом в конце. Параметр Buffer является указателем на буфер в виде строки с нулевым символом в конце, в который функция заносит имя и путь приложения, связанно­го с файлом FileName.

При успешном завершении функция FindExecutable возвращает значение, бо­льшее 32. Если возвращено меньшее значение, это свидетельствует об ошибке. Возможные значения ошибок те же, что и для приведенных ранее функций.

Пример применения функции FindExecutable.

Операторы

var APchar:array[0. .254] of char ;

FindExecutable ( ‘Exec.doc’, nil, APchar) ;

приведут к тому, что в массив APchar будет занесено имя приложения, связанного с файлом типа doc, например:

С:\MSOFFICE\WINWORD\WINWORD.ЕХЕ

**4.2.4. Создание потока**

Для создания потока используется функция CreateThread:

function CreateThread (lpThreadAttributes: Pointer; dwStackSize: DWORD;

lpStartAddress: TFNThreadStartRoutine; lpParameter: Pointer;

dwCreationFlags: DWORD; var lpThreadId: DWORD): THandle;

Функция возвращает дескриптор потока.

Входные параметры данной функции перечислены в таблице 4.5.

Таблица 4.5

Параметры функции CreateThread

|  |  |  |
| --- | --- | --- |
| Параметр | | Описание |
| lpThreadAttributes | | Указатель на структуру SECURITY\_ATTRIBUTES. Если в этом параметре указать nil, то потоку будут присвоены атрибуты защиты по умолчанию. |
| dwStackSize | | Размер стека потока. Если в этом параметре передать 0, то создается стек, размер которого указан по умолчанию в заголовке EXE-файла. |
| lpStartAddress | | Адрес функции потока, с которой должен будет начать работу создаваемый поток |
| lpParameter | | Данный параметр позволяет передавать функции потока какое-либо инициализирующее значение. |
| dwCreationFlags | Дополнительные флаги, управляющие созданием потока. Данный параметр принимает одно из двух значений: 0 (исполнение потока начинается немедленно) и CREATE\_SUSPENDED (система создает поток, инициализирует его и приостанавливает до последующих указаний) | |
| lpThreadId | Идентификатор потока. | |

# 4.2.5. Завершение процесса

Процесс завершается, когда один из его потоков вызывает процедуру ExitProcess:

procedure ExitProcess(uExitCode: UINT);

uExitCode – в эту переменную заносится код завершения процесса.

Если за вызовом этой процедуры в программе присутствует какой-либо код, он никогда не будет выполнен.

Вызов функции TerminateProcess, также завершает процесс. Данная функция имеет следующий вид:

function TerminateProcess(hProcess: THandle; uExitCode: UINT): BOOL;

где

hProcess – дескриптор завершаемого процесса

uExitCode – в данной переменной возвращается код завершения процесса.

Если функция TerminateProcess успешно выполнена, она возвращает ненулевое значение (true). Если произошла ошибка – возвращается 0 (false).

Главное отличие функции TerminateProcess от функции ExitProcess в том, что TerminateProcess может вызвать любой поток, любого процесса для завершения другого процесса.

## 4.2.6. Завершение потока

Поток можно завершить принудительно, вызвав процедуру ExitThread:

procedure ExitThread(dwExitCode: DWORD);

где в переменную dwExitCode заносится код завершения потока.

Завершить поток можно также вызвав функцию TerminateThread:

function TerminateThread(hThread: THandle; dwExitCode: DWORD): BOOL;

где hThread - дескриптор завершаемого потока

dwExitCode - в данной переменной возвращается код завершения потока.

Если функция TerminateThread успешно выполнена, она возвращает ненулевое значение (true). Если произошла ошибка – возвращается 0 (false).

В отличие от ExitThread, которая завершает поток, вызывающий данную процедуру, функция TerminateThread завершает любой поток, дескриптор которого указан в параметре hThread.

## 4.2.7. Изменение класса приоритета процесса

Для изменения класса приоритета процесса используется функция SetPriorityClass:

function SetPriorityClass(hProcess: THandle; dwPriorityClass: DWORD): BOOL;

где hProcess – дескриптор процесса

dwPriorityClass- идентификатор класса приоритета процесса.

Возможные значения переменной dwPriorityClass приведены в таблице 4.6.

Таблица 4.6.

Флаги для обозначения класса приоритета процесса

|  |  |
| --- | --- |
| Флаг | Класс приоритета |
| REALTIME\_PRIORITY\_CLASS | Real-time |
| HIGH\_PRIORITY\_CLASS | High |
| NORMAL\_PRIORITY\_CLASS | Normal |
| IDLE\_PRIORITY\_CLASS | Idle |

Если функция SetPriorityClass успешно выполнена, она возвращает ненулевое значение (true). Если произошла ошибка – возвращается 0 (false).

Например, процесс пытается установить сам себе класс приоритета idle:

Flg=SetPriorityClass(GetCurrentProcess(),IDLE\_PRIORITY\_CLASS)

## 4.2.8. Получение информации о классе приоритета процесса

Для получения информации о классе приоритета процесса применяется функция GetPriorityClass

function GetPriorityClass(hProcess: THandle): DWORD;

где hProcess – дескриптор процесса

Функция возвращает один из перечисленных в таблице 4.6 флагов.

## 4.2.9. Изменение уровня приоритета потока

Для измененияуровня приоритета потока применяется функция SetThreadPriority:

function SetThreadPriority(hThread: THandle; nPriority: Integer): BOOL;

где hThread – дескриптор потока

nPriority – идентификатор, показывающий на сколько изменяется уровень приоритета потока относительно класса. Возможные значения идентификатора приведены в таблице 4.7.

# 4.2.10. Получение информации о приоритете потока

Для получения информации о приоритете потока используется функция GetThreadPriority:

function GetThreadPriority(hThread: THandle): Integer;

где hThread – дескриптор потока.

Функция возвращает один из перечисленных в таблице 4.7. идентификаторов.

Таблица 4.7.

Идентификатор уровня приоритета потока

|  |  |
| --- | --- |
| Значение идентификатора | Описание |
| THREAD\_PRIORITY\_TIME\_CRITICAL | Time-critical. Поток выполняется c приоритетом 31 в классе real-time и с приоритетом 15 в других классах. |
| THREAD\_PRIORITY\_HIGHEST | Highest. Поток выполняется с приоритетом на два уровня выше обычного для данного класса |
| THREAD\_PRIORITY\_ABOVE\_NORMAL | Above Normal. Поток выполняется с приоритетом на один уровень выше обычного для данного класса |
| THREAD\_PRIORITY\_NORMAL | Normal. Поток выполняется с обычным уровнем приоритета для данного класса |
| THREAD\_PRIORITY\_BELOW\_NORMAL | Below Normal. Поток выполняется с приоритетом на один уровень ниже обычного для данного класса |
| THREAD\_PRIORITY\_LOWEST | Lowest. Поток выполняется с приоритетом на два уровня ниже обычного для данного класса |
| THREAD\_PRIORITY\_IDLE | Idle. Поток выполняется c приоритетом 15 в классе real-time и с приоритетом 1 в других классах. |

## 4.4. Примеры программ для работы с процессами и потоками

## 4.4.1. Создание процесса с помощью функции CreateProcess.

В данном примере создается процесс «блокнот».

VAR StartInfo:TStartUpInfo;

ProcInfo:TProcessInformation;

begin

//Очистка полей структуры StartInfo

FillChar(StartInfo,Sizeof(StartInfo),#0);

StartInfo.cb:=Sizeof(StartInfo);

CreateProcess(nil,'C:\WINNT\system32\notepad.exe C:\File1.txt',nil,nil,False,

NORMAL\_PRIORITY\_CLASS,nil,nil,StartInfo,ProcInfo);

end;

## 4.4.2. Создание процесса с помощью функцииWinExec.

VAR ErrCode:Integer;

PName:Array [0..79] of char;

begin

If OpenDialog1.Execute then

begin

ErrCode:=WinExec(StrPCopy(PName,OpenDialog1.FileName),SW\_RESTORE);

PName:=PName;

Case ErrCode of

0: ShowMessage('Не хватает памяти или ресурсов');

ERROR\_BAD\_FORMAT:

ShowMessage('Ошибочный файл '+PName);

ERROR\_FILE\_NOT\_FOUND:

ShowMessage('Не найден файл '+PName);

ERROR\_PATH\_NOT\_FOUND:

ShowMessage('Не найден каталог '+ExtractFilePath(OpenDialog1.FileName));

end;

end;

end;

## 4.4.4. Создание процесса с помощью функции ShellExecute.

VAR

POperation:Array[0..50] of Char;

PFilename:Array[0..254] of Char;

begin

POperation:='open'+#0;

PFilename:='C:\WINNT\system32\calc.exe'+#0;

ShellExecute(Handle,POperation,PFilename,nil,nil,SW\_RESTORE);

end;

## 4.4.4. Создание многопоточного приложения.

В данном примере создается многопоточное приложение, выполняющее следующие действия:

Первичный поток открывает файл для записи и запускает еще два потока. Второй поток за предоставляемый квант времени увеличивает счетчик на 1. Третий поток за предоставляемый квант времени записывает очередное значение счетчика в файл. Первичный поток работает, пока значение счетчика не станет больше 30, после чего первичный поток завершает работу второго и третьего потоков.

1. Описание глобальных переменных, используемых всеми потоками. (Например, в блоке VAR в начале программы).

var

Form1: TForm1;

sum:Integer;

f:TextFile;

implementation …..

2. В начале программы следует описать второй и третий потоки.

implementation

{$R \*.dfm}

procedure Thread1;

begin

While Sum<20 do

begin

Inc(Sum);

Sleep(30);

end;

end;

procedure Thread2;

begin

While Sum <20 do

begin

Writeln(f,IntToStr(Sum));

Sleep(30);

end;

end;

4. Первичный поток имеет вид:

VAR hThread1,hThread2:THandle;

pFunc1,pFunc2:pointer;

ThreadID1,ThreadID2:CARDINAL;

begin

Sum:=0;

AssignFile(f,'C:\temp.txt');

Rewrite(f);

pFunc1:=@Thread1;

pFunc2:=@Thread2;

hThread1:=CreateThread(nil,0,pFunc1,nil,0,ThreadID1);

hThread2:=CreateThread(nil,0,pFunc2,nil,0,ThreadID2);

While Sum<20 do Sleep(30);

ShowMessage ('Работа завершена');

CloseFile(f);

TerminateThread(hThread1,0);

TerminateThread(hThread2,0);

end;

# 4.4. Задания для самостоятельной работы

1. Программа, выполняющая запуск любого процесса и изменяющая приоритет данного процесса.
2. Программа, выполняющая запуск любого процесса и завершение данного процесса по команде пользователя.
3. Создание двух программ. Первая программа выводит на экран случайные числа. Вторая программа запускает первую как процесс и завершает работу первой программы при нажатии управляющей клавиши.
4. Программа выводит на экран содержимое каталога. При выборе любого файла последний запускается на выполнение при помощи функции CreateProcess (при условии, что выбран исполняемый файл).
5. Программа выводит на экран содержимое каталога. При выборе любого текстового файла при помощи функции CreateProcess запускается блокнот или редактор Write и открывается выбранный файл.
6. Создание многопоточной программы. Вторичный поток создается при нажатии управляющей клавиши и выводит на экран последовательность четных чисел. Работа вторичного потока завершается при нажатии управляющей клавиши.
7. Создание программы, имеющей три потока. Каждый поток рисует в окне приложения соответственно эллипс, прямоугольник и текстовую строку.
8. Создание программы, имеющей несколько потоков. Вторичные потоки создаются и завершаются по команде пользователя. Первичный поток выводит на экран время создания, завершения и работы каждого вторичного потока.
9. Создание программы, имеющей два потока. Вторичный поток выполняет любые циклические вычисления и завершается по команде пользователя. Первичный поток выводит на экран промежуточные результаты вычислений, выполняемых вторичным потоком.

## Контрольные вопросы

1. Сравните возможности функций CreateProcess? WinExec и ShellExecute
2. В чем заключается различие между процедурой ExitProcess и функцией TerminateProcess?
3. Почему ExitProcess и ExitThread реализованы как процедуры, TerminateProcess и TerminateThread – как функции?
4. Как установить приоритет текущего процесса равным 31?
5. Как установить приоритет текущего процесса равным 1?