Основы программирования на 32-битном ассемблере

Введение

Целью курсовой работы является изучение основ программирования на 32-битном ассемблере, при помощи которого необходимо разработать программу, демонстрирующую работу одного из разделов ОС WINDOWS.

В данном отчете отражена работа с потоками и процессамиWINDOWS, рассмотрены методы их создания, основанные на API-функциях. На основании описанных функций разработана программа.

1. Теоретическое обоснование

1.1 Основные сведения

Основные положения программирования в ОС WINDOWS:

· Программирование в Windows основывается на использовании функций API (Application Program Interface, т.е. интерфейс программного приложения). Их количество достигает двух тысяч. Программа в значительной степени состоит из таких вызовов. Все взаимодействие с внешними устройствами и ресурсами операционной системы происходит посредством таких функций.

· Список функций API и их описание перечислен в файле WIN32.HLP, который поставляется, например, с пакетом Borland C++.

· Главным элементом программы в среде Windows является окно. Для каждого окна определяется своя процедура обработки сообщений.

· Окно может содержать элементы управления: кнопки, списки, окна редактирования и др. Эти элементы, по сути, также являются окнами, но обладающими особыми свойствами. События, происходящие с этими элементами (и самим окном), приводят к приходу сообщений в процедуру окна.

· Операционная система Windows использует линейную модель памяти. Другими словами, всю память можно рассматривать как один сегмент. Для программиста на языке ассемблера это означает, что адрес любой ячейки памяти будет определяться содержимым одного 32-битного регистра, например EBX.

· Следствием пункта 5 является то, что фактически нет ограничений в объеме данных, кода или стека (объеме локальных переменных). Выделение в тексте программы сегмента кода и сегмента данных является теперь простой формальностью, улучшающей читаемость программы.

· Операционная система Windows является многозадачной средой.

Каждая задача имеет свое адресное пространство и свою очередь сообщений. Более того, даже в рамках одной программы может быть осуществлена многозадачность - любая процедура может быть запущена как самостоятельная задача.

1.2 Основы вызова API-функций

Функции API в ассемблере вызываются способом, схожим с вызовом в языках высокого уровня. Для этого все передаваемые параметры передаются в стек в обратном порядке.

Например, функция:

intMessageBox (HWNDhWnd, LPCTSTRlpText, LPCTSTRlpCaption, UINTuType);

будет вызвана в ассемблере по следующему шаблону:

```
includelib user32.lib

EXTERN MessageBoxA@16:NEAR

MB_OK equ 0

STR1 DB "Неверный ввод! ",0

STR2 DB "Сообщение об ошибке.",0

HW DWORD ?

PUSH MB_OK

PUSH OFFSET STR1

PUSH OFFSET STR2

PUSH HW

CALL MessageBoxA@16
```

В таком вызове API функции MessageBox добавляется символ A, указывающий на ANSI-стандарт, символ @ в роли разделителя и число N в конце, обозначающий количество байт информации, которая передается в функцию через стек.

На замену стандартному вызову при помощи команды CALL в компиляторе MASM добавлена команда INVOKE. Вызовфункции:

```
PUSH par1
```

PUSH par2

PUSH par3

PUSH par4

CALL NAME_PROC@N

Принимает вид:

INVOKE NAME_PROC, par4, par3, par2, par1

При этом функция объявляется в виде:

PROC PROTO :DWORD, :DWORD, :DWORD, :DWORD

Тогда код предыдущего примера выглядит следующим образом:

```
includelib user32.lib

MessageBoxA PROTO :DWORD, :DWORD, :DWORD, :DWORD,

MB_OK equ 0

STR1 DB "Неверный ввод! ",0

STR2 DB "Сообщение об ошибке.",0

HW DWORD ?

INVOKE MessageBoxA,HW,OFFSET STR1,OFFSET STR2,MB_OK
```

Аналогичным образом задаются структуры. Так, структуре:

typedef struct tagMSG { // msg

HWND hwnd;

UINT message;

WPARAM wParam;

LPARAM lParam;

DWORD time;

POINT pt;

} MSG;

соответствуеткод:

MSGSTRUCT STRUC

MSHWND DD ?

MSMESSAGE DD ?

MSWPARAM DD ?

MSLPARAM DD ?

MSTIME DD ?

MSPT DD ?

MSGSTRUCTENDS

1.3 Основы программирования в операционной системе Windows

Далее рассмотрена основная структура программы. В программе имеется главное окно, а следовательно, и процедура главного окна. В целом, в коде программы можно выделить следующие секции:

· Регистрация класса окон

· Создание главного окна

· Цикл обработки очереди сообщений

· Процедура главного окна

Регистрация класса окон. Регистрация класса окон осуществляется с помощью функции RegisterClassA, единственным параметром которой является указатель на структуру WNDCLASS, содержащую информацию об окне.

Создание окна. На основе зарегистрированного класса с помощью функции CreateWindowExA (или CreateWindowA) можно создать экземпляр окна. Как можно заметить, это весьма напоминает объектную модель программирования.

Цикл обработки очередей сообщений. Функция GetMessage() "отлавливает" очередное сообщение из ряда сообщений данного приложения и помещает его в структуру MSG.

Что касается функции TranslateMessage, то ее компетенция касается сообщений WM_KEYDOWN и WM_KEYUP, которые транслируются в WM_CHAR и WM_DEADCHAR, а также WM_SYSKEYDOWN и WM_SYSKEYUP, преобразующиеся в WM_SYSCHAR и WM_SYSDEADCHAR. Смысл трансляции заключается не в замене, а в отправке дополнительных сообщений. Так, например, при нажатии и отпускании алфавитно-цифровой клавиши в окно сначала придет сообщение WM_KEYDOWN, затем WM_KEYUP, а затем уже WM_CHAR.

Как можно видеть, выход из цикла ожиданий имеет место только в том случае, если функция GetMessage возвращает 0. Это происходит только при получении сообщения о выходе (сообщение WM_QUIT). Таким образом, цикл ожидания играет двоякую роль: определенным образом преобразуются сообщения, предназначенные для какого-либо окна, и ожидается сообщение о выходе из программы.

```
MSG_LOOP:

INVOKE GetMessageA,  OFFSET MSG,0,0,0

CMP AX, 0

JE END_LOOP

INVOKE TranslateMessage,OFFSET  MSG

INVOKE DispatchMessageA,OFFSET  MSG

JMP MSG_LOOP

END_LOOP:

INVOKE ExitProcess,[MSG.MSWPARAM]

_ERR:

JMP END_LOOP
```

Процедура главного окна. Описывается следующим образом:

```
WNDPROC PROC

PUSH EBP

MOV EBP, ESP ; теперь EBP указывает на вершину стека

PUSH EBX

PUSH ESI

PUSH EDI

PUSH DWORD PTR [EBP+14H]; LPARAM (lParam)

PUSH DWORD PTR [EBP+10H]; WPARAM (wParam)

PUSH DWORD PTR [EBP+0CH]; MES (message)

PUSH DWORD PTR [EBP+08H]; HWND (hwnd)

CALL DefWindowProcA@16

POP EDI

POP ESI

POP EBX

POP EBP

RET 16

WNDPROC ENDP
```

Также очень часто диалоговые окна могут предоставить замену стандартному окну, их преимущество в большей простоте реализации. К тому же при помощи сред разработки возможно создание файла ресурсов с помощью интуитивного графического редактора, что дает готовое решение дизайна диалогового окна.

Пример создания диалогового окна без использования файла ресурсов:

· Опишем прототип

```
DlgProc PROTO :DWORD,:DWORD,:DWORD,:DWORD
```

· Опишем диалоговое окно и его компоненты:

```
Dialog "Scrolling Caption in MASM32 Dialog", \

"MS Sans Serif",10, \

WS_OVERLAPPED or \
```

WS_SYSMENU or DS_CENTER, \

3, \

50,50,200,100, \

1024

DlgButton "Start thread",WS_TABSTOP,150,5,40,13,IDOK

DlgButton "Turn off", WS_TABSTOP, 150,20,40,13,IDSTOP

DlgButton "Cancel",WS_TABSTOP,150,35,40,13,IDCANCEL

DlgStatic "MASM32 Dialog",SS_LEFT,5,5,60,9,100

· ВызовокнапроисходитспомощьюCallModalDialog: CallModalDialog hInstance,0,DlgProc,NULL

Где DlgProc - функция обработчика.

1.4 Многозадачное программирование

Под процессом будем понимать объект, создаваемый операционной системой Windows обычно при загрузке исполняемого модуля и получающий в единоличное пользование:

1. Виртуальную память, выделяемую для него операционной системой.

2. Дескрипторы открываемых им файлов.

3. Список загруженных им в его собственную память динамических модулей (DLL).

4. Созданные им подпроцессы или потоки, исполняемые независимо друг от друга, в собственной памяти процесса.

Теперь что касается подпроцесса. Смысл его достаточно прост: каждый процесс в отведенном для него адресном пространстве может порождать еще процессы. Эти процессы выполняются независимо друг от друга и от порождающего их процесса. Однако порождающий процесс может при желании "убить" любой из порожденных им процессов. Такие процессы называют еще потоками, а также цепочками или нитями. Однако в операционной системе Windows для создания потоков есть и специальные средства, речь о которых пойдет ниже.

Ваше приложение может создавать процессы, запустив ту или иную EXE-программу, которые будут работать независимо от основного приложения. Одновременно Ваше приложение может при необходимости удалить запущенное им приложение из памяти. Запустить приложение (создать процесс) можно при помощи функции CreateProcess. Сейчас мы дадим описание этой функции. Ниже объясняются ее параметры.

· 1-й параметр - указывает на имя запускаемой программы. Имя может содержать полный путь к программе.

· 2-й параметр - его значение зависит от того, является первый параметр NULL (0) или нет. Если первый параметр указывает на строку, то данный параметр трактуется как командная строка запуска (без имени программы). Если первый параметр равен NULL, то данный параметр рассматривается как командная строка, первый элемент которой представляет собой имя программы. Если путь к программе не указан, то функция CreateProcess осуществляет поиск программы по определенному алгоритму:

1. Поиск в каталоге, откуда была запущена программа.

2. Поиск в текущем каталоге.

3. Поиск в системном каталоге (можно получить через GetSystemDirectory). Обычно системным каталогом является C:\WINDOWS\SYSTEM.

4. Поиск в каталоге Windows (можно получить через GetWindowsDirectory). Обычно этим каталогом является C:\WINDOWS.

5. Поиск в каталогах, перечисленных в параметре PATH окружения.

· 3-й и 4-й параметры. Используются для задания атрибутов доступа порождаемого процесса. Обычно полагают равным 0.

· 5-й параметр. Если этот параметр 0, то порождаемый процесс не наследует дескрипторы порождающего процесса, в противном случае порождаемый процесс наследует дескрипторы.

· 6-й параметр. Может менять свойства порождаемого процесса. Если параметр равен нулю, то свойства задаются по умолчанию. В силу большого количества значений этого параметра, мы отсылаем желающих к соответствующим справочным руководствам.

· 7-й параметр. Является указателем на буфер, содержащий параметры среды. Если параметр равен 0, то порождаемый процесс наследует параметры среды порождающего процесса.

· 8-й параметр. Задает текущее устройство и каталог для порождаемого процесса. Если параметр равен NULL, порождаемый процесс наследует текущее устройство и каталог порождающего процесса.

· 9-й параметр. Представляет указатель на структуру, которая содержит информацию об окне создаваемого процесса. Ниже будут рассмотрены поля этой структуры.

· 10-й параметр. Указывает на структуру, заполняемую при выполнении запуск приложения. Вот эта структура:

· PROCINF STRUC

· hProcess DD ? ; дескриптор созданного процесса.

· hThread DD ? ; дескриптор главного потока нового процесса.

· Idproc DD ? ; идентификатор созданного процесса.

· idThr DD ? ; идентификатор главного потока нового процесса.

PROCINF ENDS

Основное отличие дескриптора от идентификатора заключается в том, что дескриптор уникален лишь в пределах данного процесса, идентификатор же является глобальной величиной. Посредством идентификатора может быть найдена база данных текущего процесса. У читателя, я думаю, сразу возникнет вопрос: а чем дескриптор приложения, который мы получаем при помощи функции GetModuleHandle, от только что упомянутых величин? Дескриптор приложения, или дескриптор модуля есть величина локальная, т.е. действующая в пределах данного процесса и, как правило, равная адресу загрузки модуля в виртуальное адресное пространство. Дескриптор модуля имеется у любого модуля, загруженного в память, в том числе и у подчиненных DLL-библиотек.

Рассмотрим теперь структуру, на которую указывает 9-й параметр функции CreateProcess. Вот эта структура:

STARTUP STRUC

cb DD 0

lpReserved DD 0

lpDesktop DD 0

lpTitle DD 0

dwX DD 0

dwY DD 0

dwXSize DD 0

dwYSize DD 0

dwXCountChars DD 0

dwYCountChars DD 0

dwFillAttribute DD 0

dwFlags DD 0

wShowWindow DW 0

cbReserved2 DW 0

lpReserved2 DD 0

hStdInput DD 0

hStdOutput DD 0

```
hStdError  DD 0

STARTUP  ENDS
```

Итак, разберем смысл полей этой структуры.

cb - размер данной структуры в байтах. Заполняется обязательно.

lpReserved - резерв, должно быть равно нулю.

lpDesktop - имя рабочего стола (и рабочей станции). Имеет смысл только для Windows NT.

lpTitle - название окна для консольных приложений, создающих свое окно. Для остальных приложений должно быть равно 0.

dwX - координата X левого верхнего угла окна.

dwY - координата Y левого верхнего угла окна.

dwXSize - размер окна по X.

dwYSize - размер окна по Y.

dwXCountChars - размер буфера консоли по X.

dwYCountChars - размер буфера консоли по Y.

dwFillAttribute - начальный цвет текста. Имеет значение только для консольных приложений.

dwFlags - флаг значения полей. Вот значение этого флага.

Макро-значение флага	Значение константы	Смысл значения
STARTF_USESHOWWINDOW	1h	Разрешить поле dwShowWindow
STARTF_USESIZE	2h	Разрешить dwXSize и dwYSize
STARTF_USEPOSITI0N	4h	Разрешить dwX и dwY
STARTF_USECOUNTCHARS	8h	Разрешить dwXCountChars и dwYCountChars
STARTF_USEFILLATTR1BUTE	10h	Разрешить dwFillAttribute
STARTF_FORCEONFEEDBACK	40h	Включить возврат курсора
STARTF_FORCEOFFFEEDBACK	80h	Выключить возврат курсора
STARTF_USESTDHANDLES	100h	Разрешить hStdInput

wShowWindow - определяет способ отображения окна.

cbReserved2 - резерв, должно быть равно 0.

hStdInput - дескриптор ввода (для консоли).

hStdOutput - дескриптор вывода (для консоли).

hStdError - дескриптор вывода сообщения об ошибке (для консоли).

Поток может быть создан при помощи функции CreateThread. Рассмотрим параметры этой функции.

· 1-й параметр. Указатель на структуру атрибутов доступа. Имеет значение только для Windows NT. Обычно полагается NULL.

· 2-й параметр. Размер стека потока. Если параметр равен нулю, то берется размер стека по умолчанию, равный размеру стека родительского потока.

· 3-й параметр. Указатель на потоковую функцию, с вызова которой начинается исполнение потока.

· 4-й параметр. Параметр для потоковой функции.

· 5-й параметр. Флаг, определяющий состояние потока. Если флаг равен 0, то выполнение потока начинается немедленно. Если значение флага потока равно CREATE_SUSPENDED (4H), то поток находится в состояние ожидания и запускается по выполнению функции ResumeThread.

· 6-й параметр. Указатель на переменную, куда будет помещен дескриптор потока.

Как уже было сказано, выполнение потока начинается с потоковой функции. Окончание работы этой функции приводит к естественному окончанию работы потока. Поток также может закончить свою работу, выполнив функцию ExitThread с указанием кода выхода. Наконец, порождающий поток может закончить работу порожденного потока при помощи функции TerminateThread

Вообще идеальной мне кажется ситуация, когда функция окна берет на себя только реакцию на события, происходящие с элементами, а всю трудоемкую работу (сложные вычисления, файловая обработка) должны взять на себя потоки. Кстати, поток может создавать новые потоки, так что в результате может возникнуть целое дерево.

Для взаимодействия между потоками могут использоваться:

1. Семафоры

2. События

3. Критические секции

4. Глобальные переменные

5. Сообщения

Семафор представляет собой глобальный объект, позволяющий синхронизировать работу двух или нескольких процессов или потоков. Для программиста семафор - это просто счетчик. Если счетчик равен N, это означает, что к ресурсу имеют доступ N процессов.

Сначала при помощи функции CreateSemaphore создается семафор и его дескриптор присваивается глобальной переменной. Перед попыткой обращения к ресурсам, доступ к которым необходимо ограничить, поток должен вызвать функцию WaitForSingleObject. При открытии доступа функция возвращает 0. По окончании работы с ресурсом следует вызвать функцию ReleaseSemaphore. Тем самым увеличивается счетчик доступа на 1. С помощью семафора можно регулировать количество потоков, которые одновременно могут иметь доступ к ресурсу. Максимальное значение счетчика как раз и определяет, сколько потоков могут получить доступ к ресурсу одновременно.

События. Событие является объектом, очень похожим на семафор, но в несколько видоизмененном виде.

CreateEvent - создает объект-событие. Параметры функции.
1-й параметр. Имеет тот же смысл, что и первый параметр функции CreateSemaphore. Обычно полагается равным NULL.

2-параметр. Если параметр не равен нулю, то событие может быть сброшено при помощи функции ResetEvent. Иначе событие сбрасывается при доступе к нему какого либо процесса.

3-й параметр. Если параметр равен 0, то событие инициализируется как сброшенное, в противном случае сразу же подается сигнал о наступлении соответствующей ситуации.

4-й параметр. Указатель на строку, которая содержит имя события.

Ожидание события осуществляется, как и в случае с семафором, функцией WaitForSingleObject.

Критические секции. Понятие критической секции позволяет уберечь определенные области программы так, чтобы в этой области программы в данный момент времени исполнялся бы только один поток. Рассмотрим функции для работы с критической секцией.

InitializeCriticalSection - данная функция создает объект под названием критическая секция. Параметры функции.

· 1-й параметр. Указатель на структуру, указанную ниже. Поля данной структуры используются только внутренними процедурами, и смысл их безразличен.

· CRITICAL_SECTION STRUCT

· DebugInfo DWORD ?

· LockCount LONG ?

· RecursionCount LONG ?

· OwningThread HANDLE ?

· LockSemaphore HANDLE ?

· SpinCount DWORD ?

CRITICAL_SECTION ENDS

EnterCriticalSection - войти в критическую секцию.

После выполнения этой функции данный поток становится владельцем данной секции. Следующий поток, вызвав данную функцию, будет находиться в состоянии ожидания. Параметр функции такой же, что и в предыдущей функции.

LeaveCriticalSection - покинуть критическую секцию. После этого второй поток, который был остановлен функцией EnterCriticalSection, станет владельцем критической секции. Параметр функции LeaveCriticalSection такой же, как и у предыдущих функций.

DeleteCriticalSection - удалить объект "критическая секция". Параметр аналогичен предыдущим.

Программно можно определить несколько объектов критической секции, с которыми будут работать несколько потоков. Два потока обращаются время от времени к процедуре, выводящей очередной символ из строки в окно. В результате такой конкурентной деятельности должна быть напечатана строка.

Часть процедуры, выводящей очередной символ, сделана критической, поэтому доступ к выводу в окно в данный момент времени имеет только один поток.

2. Демонстрационная программа

2.1 Листинг

```
; ««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««

.486 ;

.model flat, stdcall ;

option casemap :none ;

; Подключаем файлы

; ~~~~~~~~~~~~~~

include ..\include\windows.inc

include ..\include\user32.inc

include ..\include\kernel32.inc

include ..\macros\macros.asm

include ..\include\masm32.inc

; Библиотеки
```

```
; ~~~~~~~~~~
includelib  ..\lib\user32.lib
includelib  ..\lib\kernel32.lib
includelib  ..\lib\masm32.lib
include  ..\include\dialogs.inc
; Описание прототипов
DlgProc  PROTO  :DWORD,:DWORD,:DWORD,:DWORD
rotate_caption  PROTO :DWORD
; Дата сегмент
.data
textBufferCaption  db "Потоков  было запущенно",0
prog db "c:\windows\notepad.exe",0
prog1 db "c:\windows\regedit.exe",0
processInfo  PROCESS_INFORMATION <>
.data?
hWnd  dd ?
hhwn  dd ?
hInstance  dd ?
ThreadID  DWORD  ?
threadSwitch  dd ?
threadCount  dd ?
textBuffer  db 20 dup(?)
startInfo  dd ?
; Константы
.const
IDSTOP equ 102
```

```
IDPROC equ 103
IDPROC1 equ 104
IDOK1 equ 105
IDOK2 equ 107
WM_STOP_THREAD equ WM_USER+1
WM_STOP_THREAD2 equ WM_USER+1
; «««««««««««««««««««««««««««««««КОД«««««««««««««««««««««««««««««««««
.code
start:
mov hInstance, FUNC(GetModuleHandle,NULL)
;Описываем диалог
Dialog "Курсовая работа по ОС, выполнил Григорьев Ю.Н. 8-78-3", \ ;
"MS Sans Serif",10, \ ; Шрифт и его размер
WS_OVERLAPPED or \ ; Стиль
WS_SYSMENU or DS_CENTER, \ ; диалогового окна
7, \ ; Число элементов управляения
50,50,200,100, \ ; x y координаты
1024 ; Размер буффера
;Описываем кнопки
DlgButton "Start thread",WS_TABSTOP,150,5,40,13,IDOK
DlgButton "End thread",WS_TABSTOP,100,5,50,13,IDOK1
DlgButton "End thread2",WS_TABSTOP,100,20,50,13,IDOK2
DlgButton "Thr. started", WS_TABSTOP, 150,20,40,13,IDSTOP
DlgButton "Start note", WS_TABSTOP, 150,35,40,13,IDPROC
DlgButton "Start regedit", WS_TABSTOP, 150,50,40,13,IDPROC1
DlgButton "Cancel", WS_TABSTOP,150,65,40,13,IDCANCEL
```

```
;Вызов окна
CallModalDialog  hInstance,0,DlgProc,NULL
; ««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««
;Процедура обработчика диалогового окна
DlgProc  proc hWin:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD
;Разбор сообщения
Switch uMsg
Case WM_INITDIALOG  ; на создании
m2m hWnd, hWin  ; копируем дескриптор
mov threadCount,  0 ;счетчик потоков = 0
Case WM_COMMAND ; разбор сообщений кнопок
Switch wParam
Case IDOK
mov threadSwitch,0  ; маска для работы потоков
;Если = 0 то работа разрешена
mov eax,OFFSET  thread_proc1  ; в регистр адрес процедуры потока
invoke  CreateThread,NULL,NULL,eax,NULL,NULL,ADDR  ThreadID  ; вызываем поток
;;;
Case IDOK1
mov eax,OFFSET  thread_proc3  ; вызов второго вида потока
invoke  CreateThread,NULL,NULL,eax,NULL,NULL,ADDR  ThreadID
Case IDOK2
mov eax,OFFSET  thread_proc4  ; вызов второго вида потока
invoke  CreateThread,NULL,NULL,eax,NULL,NULL,ADDR  ThreadID
;;;
Case IDCANCEL  ;
```

```
jmp quit_dialog ; закрыть диалог

Case IDSTOP ;остановка потоков

mov eax,OFFSET thread_proc2 ; вызов второго вида потока

invoke CreateThread,NULL,NULL,eax,NULL,NULL,ADDR ThreadID

Case IDPROC ; создание процесса

invoke GetStartupInfo,ADDR startInfo ; получаем информацию

;вызываем процесс

invoke CreateProcess,NULL,offset
prog,NULL,NULL,FALSE,NORMAL_PRIORITY_CLASS,NULL,NULL,ADDR
startInfo,ADDR processInfo

Case IDPROC1; создание процесса

invoke GetStartupInfo,ADDR startInfo ; получаем информацию

;вызываем процесс

invoke CreateProcess,NULL,offset
prog1,NULL,NULL,FALSE,NORMAL_PRIORITY_CLASS,NULL,NULL,ADDR
startInfo,ADDR processInfo

EndSw

Case WM_STOP_THREAD

mov threadSwitch,1 ; по данному сообщению маска для потоков = 1

Case WM_STOP_THREAD2

invoke TerminateThread,IDOK,12 ; завершаем потоки 2 методом

Case WM_CLOSE

quit_dialog:

invoke EndDialog,hWin,0 ; закрываем диалог

EndSw

return 0

DlgProc endp

; процедура первого вида потока
```

```
; она будет крутить (скроллить) заголовок
thread_proc1 proc
;локальные переменные
LOCAL tlen :DWORD
LOCAL buffer[128]:BYTE
Local pbuf :DWORD
Local charCount :DWORD
mov charCount,0
mov pbuf, ptr$(buffer)
add threadCount, 1
@loop:
invoke GetWindowText,hWnd,pbuf,128 ; берем текст
invoke rotate_caption,pbuf;крутим на 1 символ
invoke SetWindowText,hWnd,pbuf;обновляем текст
invoke Sleep, 50;заддержка
.if threadSwitch == 1;если запрет
jmp @stopThread;выход из цикла
.endif
jmp @loop
@stopThread:
ret
thread_proc1 endp
; процедура второго вида потока
thread_proc2 proc
invoke dwtoa,threadCount,ADDR textBuffer ; число в строку
invoke MessageBox,hWnd,ADDR textBuffer,ADDR textBufferCaption,MB_OK ; выводим сколько потоков запущенно
```

```
invoke  SendMessage,hWnd,NULL,NULL,NULL  ;
ret
thread_proc2  endp
;
thread_proc3  proc
invoke  SendMessage,hWnd,WM_STOP_THREAD,NULL,NULL  ; отсылаем сообщение
mov threadCount,0  ; вычитаем
ret
thread_proc3  endp
thread_proc4  proc
invoke  SendMessage,hWnd,WM_STOP_THREAD2,NULL,NULL
mov threadCount,0  ; вычитаем
ret
thread_proc4  endp
; ««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««
OPTION PROLOGUE:NONE
OPTION EPILOGUE:NONE
; процедура прокрутки  на 1 символ
rotate_caption  proc pbuf:DWORD
mov edx,  [esp+4] ; грузим  строку
mov cl,  [edx] ; первый байт
add edx,  1 ; переходим к следующему
@rewrite:
mov al,  [edx] ; читаем символ
add edx,  1 ; увеличиваем аддрес
test al,  al ; проверяем на 0
```

```
jz @last_rewrite  ; если конец строки то на выход

mov [edx-2], al ; запись символа

jmp @rewrite

@last_rewrite:

mov [edx-2], cl ; запись последнего символа

mov BYTE PTR [edx-1], 0 ; нуль терминатор

ret 4 ; баланс стека

rotate_caption endp

OPTION PROLOGUE:PrologueDef

OPTION EPILOGUE:EpilogueDef

; ««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««

end start
```

секция функция идентификатор windows

2.2 Демонстрация работы

Вывод

В ходе данной курсовой работы были освещены основы программирования для WINDOWS. Также были изучены методы и средства многозадачного программирования в WINDOWS. Для реализации программы, иллюстрирующей теоретическую информацию был использован 32-битный ассемблер.