№ 4147

004
У 677

УПРАВЛЕНИЕ РЕСУРСАМИ В ОС WINDOWS

Методические указания

НОВОСИБИРСК
2012

Министерство образования и науки Российской Федерации

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

004
У 677

№ 4147

УПРАВЛЕНИЕ РЕСУРСАМИ В ОС WINDOWS

Методические указания
к выполнению индивидуальной работы
для студентов III курса ФПМИ
по курсу «Управление ресурсами в вычислительных системах»

НОВОСИБИРСК
2012

УДК 004.451.9(07)
У 677

Составители:
Стасышин В.М., канд. техн. наук, доц.
Куликов И.М., канд. физ.-мат. наук, доц.

Рецензент
Н.Л. Долозов, канд. техн. наук, доц.

Работа подготовлена на кафедре
программных систем и баз данных

1. ОПИСАНИЕ И ЦЕЛИ ИНДИВИДУАЛЬНОЙ РАБОТЫ

Индивидуальная работа (далее ИР) по курсу «Управление ресурсами в вычислительных системах» ставит задачей практическое освоение вопросов управления ресурсами операционных систем (далее ОС) семейства Windows. Целью ИР является изучение основных особенностей ОС Windows: работа с потоками, организация графического интерфейса, механизма обработки сообщений, использование динамически подключаемых библиотек (dll), низкоуровневое взаимодействие с процессором.

Индивидуальная работа включает четыре уровня сложности. Студент самостоятельно принимает решение, в рамках какого уровня он будет выполнять индивидуальную работу.

В рамках I уровня необходимо реализовать консольное Windows-приложение, в котором создается один дочерний поток, реализующий требования варианта задания и выводящий результат на консоль.

В рамках II уровня необходимо реализовать минимальное графическое Windows-приложение, в котором при создании окна, но до момента его отображения на экране, создается дочерний поток, который реализует задачу в соответствии с выбранным вариантом, после чего полученный результат отображается в графическом окне.

При выполнении задания на III уровне необходимо реализовать все требования II уровня с той разницей, что функция, выполняющая задание, должна быть реализована в виде динамической библиотеки.

IV уровень предполагает полную реализацию требований III уровня, но дополнительно в функции из динамической библиотеки используется ассемблерная вставка, содержание которой зависит от номера выполняемого варианта задания. Полученная информация из ассемблерной вставки выводится вместе с результатом решения задачи в соответствии с вариантом задания.

В представленных методических указаниях описываются программные средства, необходимые для реализации каждого уровня сложности. Для IV уровня сложности для варианта, аналогичного перечисленным ниже вариантам заданий, приведен полный текст основной программы и динамической библиотеки с описанием всех технологий, используемых в индивидуальной работе.

2. ОСОБЕННОСТИ ПРОГРАММИРОВАНИЯ В ОС WINDOWS

Основной особенностью программирования в ОС Windows является его объектно-ориентированность. При этом не имеет значения, использованы или не использованы для этой цели объектно-ориентированные языки. Система Windows представляет собой набор объектов, которые используются и при работе, и при программировании.

Одним из наиболее важных объектов является окно. Окно первым появляется при запуске приложения и последним исчезает при завершении работы. Общение с объектом-окном возможно посредством сообщений. Механизм сообщений в Windows реализуется при помощи очереди сообщений, поддерживаемой системой. Каждое сообщение попадает в такую очередь, где и ожидает последующей обработки. Механизм сообщений является единственным способом связи приложений со своими окнами. Сообщения представляют собой события, на которые при необходимости должно реагировать приложение.

Каждое сообщение связано с конкретным окном, с которым, в свою очередь, связана собственная оконная процедура (процедура обработки сообщений). Все сообщения, источником которых служат аппаратные средства, система помещает в первичную входную очередь. При этом сообщение после попадания в первичную входную очередь перенаправляются в частную очередь конкретного потока приложения. В самом начале работы приложение имеет единственную очередь сообщений, которая относится к первичному потоку. Если процесс заводит новый поток, которому очередь сообщений не нужна, то система не создает новую очередь сообщений. Сообщения, находящиеся в очереди, обрабатываются процедурой обработки сообщений, которая связана с каждым окном.

В настоящее время в системе Windows имеется около 1000 стандартных сообщений. Все эти сообщения можно разделить на 8 категорий:

- аппаратные сообщения;
- сообщения об организации окна;
- сообщения об организации интерфейса пользователя;
- сообщения о завершении работы;
- частные сообщения;

– уведомления о системных ресурсах;
– обмен данными;
– внутрисистемные сообщения.

Система Windows предоставляет приложению богатый набор графических устройств, которые используются для организации вывода. К ним относятся:

– битовые образы – прямоугольные массивы точек, формирующие растровые изображения;
– карандаши – используются для параметров рисования линий;
– кисти – используются для параметров заливки замкнутых контуров;
– шрифты – используются для задания параметров вывода текста;
– регионы – область окна, которая может иметь в общем случае любую форму;
– логические палитры – обеспечивают связь между приложением и устройством вывода, содержат список цветов необходимых приложению;
– контуры – используются для заполнения или выделения контура фигур.

Еще одна особенность Windows – независимость устройств: для вывода на конкретное устройство приложение использует драйвер устройства, причем на логическом уровне приложение работает с контекстом устройства, в котором хранятся все параметры вывода. Контекст устройства – структура, определяющая набор графических объектов и связанных с ними атрибутов и графических режимов, которые воздействуют на вывод. В Windows определены 4 типа контекстов устройств:

– экран;
– принтер;
– объект в памяти;
– информационный.

В рамках выполнения индивидуальной работы предполагается использовать только контекст экрана. Параметры цвета, палитры и шрифтов окна, используемого для вывода, могут быть определены по умолчанию. Сигналом для начала рисования служит, как правило, сообщение WM_PAINT, но иногда возможно немедленное рисование, не дожидаясь обработки данного сообщения.

Используя контекст устройств программист может выполнять следующие операции:

- перечисление графических объектов;
- выбор новых графических объектов;
- удаление графических объектов;
- сохранение свойств графических объектов;
- восстановление графических объектов.

3. ВАРИАНТЫ ЗАДАНИЙ И ПРАВИЛА ИХ ВЫБОРА

В табл. 3.1 приведены варианты для выполнения задания при разработке программы I, II, III уровня сложности, а также дополнительное задание для получения сведений в случае реализации IV уровня сложности.

Таблица 3.1

№	Задание для выполнения I, II, III уровня сложности	Дополнительное задание для выполнения IV уровня сложности
1	Определить приоритет текущего потока	Определить наличие сопроцессора
2	Определить приоритет текущего процесса	Определить наличие поддержки виртуального режима работы процессора
3	Определить количество мониторов	Определить наличие поддержки команды RDTSC
4	Определить количество кнопок мыши	Определить наличие команды сравнения и обмена 8 битного значения
5	Определить максимальную ширину окна (в пикселях)	Определить наличие поддержки MMX технологии
6	Определить максимальную высоту окна (в пикселях)	Определить наличие поддержки технологии SSE
7	Определить ширину экрана (в пикселях)	Определить наличие поддержки технологии SSE2
8	Определить высоту экрана (в пикселях)	Определить наличие поддержки технологии SSE3
9	Определить наличие сетевого подключения	Определить наличие поддержки технологии Hyper Threading
10	Определить число процессоров (ядер)	Определить размер страницы буфера ассоциативной трансляции команд*

Окончание табл. 3.1

№	Задание для выполнения I, II, III уровня сложности	Дополнительное задание для выполнения IV уровня сложности
11	Определить размер страницы памяти	Определить ассоциативность и количество входов буфера ассоциативной трансляции команд*
12	Определить тип процессоров (ядер)	Определить размер страницы буфера ассоциативной трансляции данных*
13	Определить текущую директорию	Определить ассоциативность и количество входов буфера ассоциативной трансляции данных*
14	Определить имя компьютера	Определить размер КЭШа данных первого уровня*
15	Определить имя пользователя	Определить ассоциативность КЭШа данных первого уровня*
16	Определить системную директорию	Определить длину строки КЭШа данных первого уровня*
17	Определить директорию Windows	Определить размер КЭШа команд первого уровня*
18	Определить текущий год	Определить ассоциативность КЭШа команд первого уровня*
19	Определить текущий день	Определить длину строки КЭШа команд первого уровня*
20	Определить текущий день недели	Определить размер КЭШа данных второго уровня*
21	Определить текущий месяц	Определить ассоциативность КЭШа данных второго уровня*
22	Определить текущий час	Определить длину строки КЭШа данных второго уровня*
23	Определить текущую минуту	Определить размер КЭШа данных третьего уровня*
24	Определить тип клавиатуры	Определить ассоциативность КЭШа данных третьего уровня*
25	Определить количество функциональных клавиш клавиатуры	Определить длину строки КЭШа данных третьего уровня*

* Выделены усложненные варианты заданий IV уровня.

Для выполнения индивидуального задания студент выбирает вариант, сообщая его номер преподавателю, ведущему лабораторные занятия по курсу в группе.

Замечание. Для студентов одной группы выбранные номера вариантов не должны повторяться.

4. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, НЕОБХОДИМОЕ ДЛЯ ВЫПОЛНЕНИЯ ИНДИВИДУАЛЬНОЙ РАБОТЫ. ФУНКЦИИ WINDOWS API

Необходимое программное обеспечение для выполнения индивидуальной работы (Visual Studio 2008, справочная система MSDN и пр.) установлено в терминальных классах факультета. Указанное программное обеспечение Microsoft студент может самостоятельно получить в Личном кабинете студента на портале университета http://www.nstu.ru.

На сегодняшний день Visual Studio 2008 является самой распространенной средой разработки и отладки приложений (версия 2010 г. пока имеет ряд недостатков, которые будут, очевидно, исправлены в ближайших версиях). Основы разработки и отладки приложений в среде Visual Studio 2008 изложены в методических указаниях [7].

Для выполнения выбранного варианта задания при разработке программы I, II, III уровня используется одна из функций Windows API, вызванная с определенным параметром. В табл. 4.1 приведены функции и необходимые параметры и/или дополнительные функции, необходимые для реализации соответствующего варианта задания. Подробное описание функций и параметров можно найти в справочной системе MSDN, установленной в терминальных классах факультета.

В качестве параметров перечисленных в таблице функций выступают указанные константы или отдельные поля указанных структур.

Таблица 4.1

№	Необходимая функция	Дополнительные функции и/или параметры
1	GetThreadPriority	GetCurrentThread
2	GetPriorityClass	GetCurrentProcess
3	GetSystemMetrics	SM_CMONITORS
4	GetSystemMetrics	SM_CMOUSEBUTTONS
5	GetSystemMetrics	SM_CXFULLSCREEN
6	GetSystemMetrics	SM_CYFULLSCREEN
7	GetSystemMetrics	SM_CXSCREEN
8	GetSystemMetrics	SM_CYSCREEN
9	GetSystemMetrics	SM_NETWORK (первый бит полученного значения)
10	GetSystemInfo	Поле dwNumberOfProcessors структуры SYSTEM_INFO
11	GetSystemInfo	Поле dwPageSize структуры SYSTEM_INFO
12	GetSystemInfo	Поле dwProcessorType структуры SYSTEM_INFO
13	GetCurrentDirectory	Текстовая строка
14	GetComputerName	Текстовая строка
15	GetUserName	Текстовая строка
16	GetSystemDirectory	Текстовая строка
17	GetWindowsDirectory	Текстовая строка
18	GetSystemTime	Поле wYear структуры SYSTEMTIME
19	GetSystemTime	Поле wDay структуры SYSTEMTIME
20	GetSystemTime	Поле wDayOfWeek структуры SYSTEMTIME
21	GetSystemTime	Поле wMonth структуры SYSTEMTIME
22	GetSystemTime	Поле wHour структуры SYSTEMTIME
23	GetSystemTime	Поле wMinute структуры SYSTEMTIME
24	GetKeyboardType	0
25	GetKeyboardType	2

Ниже описаны две структуры, содержащие информацию о системе (листинг 4.1) и времени (листинг 4.2).

Листинг 4.1

```
1:  typedef struct _SYSTEM_INFO {
2:   union {
3:       DWORD dwOemId;
4:       struct {
5:                WORD wProcessorArchitecture;
6:                WORD wReserved;
7:             };
8:        };
9:   DWORD dwPageSize;
10:  LPVOID lpMinimumApplicationAddress;
11:  LPVOID lpMaximumApplicationAddress;
12:  DWORD dwActiveProcessorMask;
13:  DWORD dwNumberOfProcessors;
14:  DWORD dwProcessorType;
15:  DWORD dwAllocationGranularity;
16:  WORD wProcessorLevel;
17:  WORD wProcessorRevision;
18:  } SYSTEM_INFO, *LPSYSTEM_INFO;
```

Следует обратить внимание на поля структуры в строках 5 (архитектура процессора), 9 (размер страницы памяти), 13 (количество процессоров доступных системе; в данном случае процессор понимается как физически независимый процессор, так и ядро, и многопотоковое устройство, например, как в случае технологии HyperThreading у процессоров Intel), 14 (тип процессора).

Листинг 4.2

```
1:  typedef struct _SYSTEMTIME {
2:       WORD wYear;
3:       WORD wMonth;
4:       WORD wDayOfWeek;
5:       WORD wDay;
6:       WORD wHour;
7:       WORD wMinute;
8:       WORD wSecond;
9:       WORD wMilliseconds;
10: } SYSTEMTIME;
```

Поля последней приведенной структуры являются английским переводом интервалов времени.

5. ВЫПОЛНЕНИЕ ИР I УРОВНЯ. РАБОТА С ПОТОКАМИ В РАМКАХ КОНСОЛЬНОГО ПРИЛОЖЕНИЯ WINDOWS

Все современные серверы и почти все настольные компьютеры с системой Windows построены на многоядерных архитектурах с общим полем памяти. Вопросы программирования многоядерных архитектур подробно будут рассмотрены в курсе «Основы параллельного программирования». Основной целью данной индивидуальной работы является получение начальных знаний и умений по созданию, запуску и завершению потоков, а также написанию функций, исполняемых в рамках созданного потока. Для большинства приложений таких знаний вполне достаточно, чтобы реализовать логику работы программы с использованием многоядерности.

Для выполнения ИР I уровня необходимо:

– разработать консольное приложение, в рамках которого создается присоединенный поток;

– разработать функцию, которая запускается в рамках созданного потока и которая реализует требования выбранного варианта задания.

Обе функции размещаются в одном файле.

Фрагмент кода консольного приложения, описывающий создание, запуск и синхронизацию потока, приведен ниже в листинге 5.1.

Листинг 5.1

```
1: DWORD WINAPI ThreadFunc(void *);
2: HANDLE  hThread;
3: DWORD   IDThread;
……….
4: hThread = CreateThread(NULL, 0, ThreadFunc, NULL, 0, &IDThread);
5: WaitForSingleObject(hThread, INFINITE);
6: CloseHandle(hThread);
```

В 1 строке листинга 5.1 приведен прототип функции-потока (предполагается, что функция ***ThreadFunc*** выполняет действия, определенные вариантом задания). Тип возвращаемого значения не так важен и может быть пустым. Обязательным является тип входного параметра. С помощью безымянного указателя можно передать аргументы для функции потока.

В строках 2 и 3 описаны дескриптор и идентификатор потока.

В строке 4 выполняется создание потока, у которого первый параметр определяет атрибуты защиты создаваемого потока и, как правило, задается значением ***NULL***. Второй параметр указывает начальный размер дополнительного стека для потока, который также игнорируется в случае установок размеров стека компилятором. Третий параметр – функция потока, прототип которой указан в первой строке. Поскольку параметры в функцию не передаются, то четвертый параметр установлен в ***NULL***, в случае передачи аргументов задается указатель на область памяти аргументов. Все операции по преобразованию типов в этом случае берет на себя программист. Пятым аргументом является флаг управления потоком. В случае, если он установлен в 0, поток запускается немедленно. Последним (выходным) параметром является системный идентификатор созданного потока. Функция ***CreateThread*** возвращается дескриптор потока.

В 5 строке листинга реализовано ожидание завершения потока. Поток идентифицируется дескриптором и бесконечным временем ожидания.

В 6 строке происходит удаление дескриптора потока.

Подробное описание Windows-потоков можно найти в [1, 2, 8].

6. ВЫПОЛНЕНИЕ ИР II УРОВНЯ. СОЗДАНИЕ БАЗОВОГО ГРАФИЧЕСКОГО ИНТЕРФЕЙСА WINDOWS И РАБОТА С ПОТОКАМИ

Как отмечалось выше, программирование для Windows является объектно-ориентированным. Одним из наиболее важных объектов является окно. Окно первым появляется при запуске приложения и последним исчезает при завершении работы.

Общение с объектом-окном возможно при помощи сообщений [3,4]. Механизм сообщений в Windows реализуется при помощи очереди сообщений, поддерживаемой системой. Каждое сообщение попадает в такую очередь, где и ожидает последующей обработки. Механизм сообщений является единственным способом связи приложений со своими окнами. Сообщения представляют собой события, на которые при необходимости должно реагировать приложение.

Каждое сообщение связано с конкретным окном, с каждым из которых, в свою очередь, связана собственная оконная процедура (про-

цедура обработки сообщений). Все сообщения, источником которых служат аппаратные средства, система помещает в первичную входную очередь. При этом сообщение после попадания в первичную входную очередь, перенаправляются в частную очередь конкретного потока приложения. В самом начале работы приложение имеет единственную очередь сообщений, которая относится к первичному потоку, если процесс заводит новый поток, которому очередь сообщений не нужна, то система не создает новую очередь сообщений. Сообщения, находящиеся в очереди, обрабатываются процедурой обработки сообщений, которая связана с каждым окном.

В рамках ИР II уровня необходимо реализовать минимальное графическое Windows-приложение, в котором при создании окна, но до момента его отображения на экране, создается дочерний поток, который реализует задачу в соответствии с вариантом, после чего полученный результат отображается в графическом окне.

Для выполнения ИР II уровня необходимо:

– разработать главную функцию ***WinMain***, в которой создается основное окно программы и запускается цикл обработки сообщений;

– разработать оконную процедуру, обеспечивающую обработку сообщений для основного окна программы;

– разработать функцию, которая запускается в рамках созданного потока в процедуре обработки сообщений (при обработке сообщения ***WM_CREATE***) и реализует требования выбранного варианта задания.

Все три перечисленные функции размещаются в одном файле.

Структура главной функции ***WinMain*** включает следующие этапы:

– регистрация класса окна;

– создание главного окна приложения;

– цикл обработки сообщений.

Любое окно Windows принадлежит одному из существующих в данный момент в системе классу, который должен быть определен до отображения окна на экране. Класс окна задает основные свойства окон, например форму курсора, меню и пр. Другими словами, класс окна – это шаблон, в котором определяются выбранные стили, шрифты, заголовки, пиктограммы, размер. При этом следует учитывать, что имя класса окна доступно всем пользователям в системе и, как следствие, должно быть уникальным, чтобы не допустить возникновения конфликтов с классами окон других приложений. После регистрации класса необходимо создать окно, для этого используется функция ***CreateWindow***.

Стандартный цикл обработки сообщений представляет собой следующий порядок действий:

– получение сообщения;
– преобразование виртуальных кодов клавиш в ASCII-значения;
– посылка сообщения в нужную оконную процедуру.

В следующем листинге 6.1 приведем пример главной функции ***WinMain*** Windows-приложения с подробными комментариями.

Листинг 6.1

```
1: int WINAPI WinMain(HINSTANCE hThisInst, HINSTANCE hPrevInst,
2:                    LPSTR str, int nWinMode)
3: {
4:  MSG msg;
5:        WNDCLASS wcl;
6:        HWND hWnd;
7:        wcl.hInstance = hThisInst;
8:        wcl.lpszClassName = szClassName;
9:        wcl.lpfnWndProc = WindowFunc;
10:       wcl.style = CS_HREDRAW | CS_VREDRAW;
11:       wcl.hIcon   = LoadIcon(NULL,IDI_APPLICATION);
12        wcl.hCursor = LoadCursor(NULL,IDC_ARROW);
13:       wcl.lpszMenuName = NULL;
14: wcl.cbClsExtra = 0;
15:       wcl.cbWndExtra = 0;
16:       wcl.hbrBackground                                    =
(HBRUSH)GetStockObject(WHITE_BRUSH);
17:       RegisterClass(&wcl);
18:       hWnd = CreateWindow(szClassName, szTitle,
19:       WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN |
20:       WS_CLIPSIBLINGS, 100, 50, 700, 120,
21:       HWND_DESKTOP, NULL, hThisInst, NULL);
22:       ShowWindow(hWnd,nWinMode);
23:       UpdateWindow(hWnd);
24:       while(GetMessage(&msg,NULL,0,0))
25: {
26:       TranslateMessage(&msg);
27:       DispatchMessage(&msg);
28:       }
29:       return msg.wParam;
30: }
```

В строках 1 и 2 в функции ***WinMain*** передаются дескрипторы текущей копии приложения и предыдущей, который всегда равен нулю, командную строку и способ отображения приложения.

В строках 4–6 описаны структуры и типы сообщений, класса окна и дескриптора окна.

В строках 7–16 выполнено создание класса окна. В строке 7 указывается дескриптор приложения, которое создает класс окна. Дескриптор устанавливается как дескриптор текущего приложения. В 8 строке указатель настраивается на строку – название класса окна приложения. Строка с названием может быть описана как LPCSTRszClassName = "Name". В строке 9 в классе окна определяется функция-обработчик сообщений (в примере предполагается, что имя этой функции – ***WindowFunc***), в которую будут поступать сообщения от системы. В строке 10 указывается стиль окна. В данном случае указано, что окно будет перерисовываться, как только изменится один из его размеров. В строках 11 и 12 указываются пиктограмма и курсор окна. Строка 13 определяет, что меню в окне не предусмотрено. Строки 14 и 15 отвечают за дополнительные параметры. В строке 16 указывается цвет фона окна.

В строке 17 происходит регистрация класса окна.

В строках 18–21 выполняется создание самого окна на базе созданного класса окна. Первые два строковых параметра функции ***CreateWindow*** – имя класса окна и название окна, которые отображаются в заголовке окна. Далее указываются параметры окна, размеры окна, дескриптор окна-родителя (в данном случае Рабочего стола), указатель на меню (в данном случае ***NULL***), дескриптор данного приложения и указатель на дополнительные данные.

Подробно параметры окна описаны в справочной системе MSDN и литературе, например, [3, 4].

В строках 22 и 23 выполняется отображение окна.

В строках 24-28 записан цикл обработки сообщений. Стандартный цикл обработки сообщений представляет собой следующие действия:

– получение сообщения (строка 24);

– преобразование виртуальных кодов клавиш в ASCII-значения (строка 26);

– посылка сообщения в нужную оконную процедуру (строка 27).

Оконная процедура обработки сообщений для основного окна программы обеспечивает обработку всех необходимых сообщений. Как

правило, процедура обработки сообщений включает оператор ***Switch***, в котором последовательно обрабатываются сообщения.

Типичная процедура обработки сообщений приведена ниже в листинге 6.2.

Листинг 6.2

```
1:  LRESULT CALLBACK WindowFunc(HWND hWnd, UINT msg,
2:                      WPARAM wParam, LPARAM lParam)
3:  {
4:   PAINTSTRUCT ps;
5:          HDC hDC;
6:          switch(msg) {
7:          case WM_CREATE:
8:    /*  создание потока, запуск процедуры в рамках созданного потока,  */
                /*  ожидание завершения,  уничтожение дескриптора потока */
9:                      break;
10:         case WM_DESTROY:
11:         PostQuitMessage(0);
12:         break;
13:         case WM_PAINT:
14:         hDC = BeginPaint(hWnd, &ps);
15:         TextOut (hDC, 10, 10, Info, strlen(Info));
16:         EndPaint(hWnd, &ps);
17:                 break;
18:         default:
19:         return DefWindowProc(hWnd,msg,wParam,lParam);
20:         }
21:         return 0;
22:  }
```

В строках 1 и 2 в оконную процедуру передаются идентификатор окна, код сообщения и два дополнительных параметра для сообщения. В строках 4 и 5 указаны структура для рисования и контекст устройства.

В листинге 6.2 обрабатываются только три сообщения: создание окна, уничтожение окна и перерисовка окна. Все остальные должны быть обработаны по умолчанию ***DefWindowProc***, в частности, это необходимо для обработки внутренних сообщений Windows.

В строке 8 при обработке сообщения ***WM_CREATE*** (оконная процедура получает сообщение ***WM_CREATE***, когда окно уже создано, но

еще не появилось на экране) создается поток, в рамках которого исполняется некоторая функция, реализующая требования варианта задания.

После завершения потока освобождаются используемые им ресурсы.

В строках 10–12 обрабатывается сообщение ***WM_DESTROY***, связанное с закрытием окна.

В строках 13–17 при обработке сообщения ***WM_PAINT*** выполняются инициализация контекста устройства, в данном случае окна, вывод строки текста ***Info***, содержащей результаты выполнения функции потока и завершение рисования.

При этом предполагается, что завершившаяся функция, которая исполнялась в рамках созданного потока и которая реализовала требования выбранного варианта задания, вернула результаты своей работы в строку текста ***Info***.

В целом процедура является стандартной, сведения об обработке сообщений с использованием карт сообщений подробно изложены, например, в [4].

Замечание. Структура оконной Windows-программы в достаточной степени является типовой. Тем не менее, при выполнении индивидуальной работы студент должен продемонстрировать понимание используемых им элементов структур, классов и пр. Это может проявляться, например, в выборе размера окна, цветовой гаммы использовании других, более сложных конструкций Windows-программирования.

7. ВЫПОЛНЕНИЕ ИР III УРОВНЯ. РАБОТА С ДИНАМИЧЕСКИ ПОДКЛЮЧАЕМЫМИ БИБЛИОТЕКАМИ

В рамках ИР III уровня необходимо реализовать графическое Windows-приложение, выполняющее все требования II уровня, но функция, выполняющая задание из предлагаемого варианта должна быть реализована в виде динамической библиотеки.

Динамически подключаемые библиотеки являются одной из особенностей ОС семейства Windows. Существует достаточно много спо-

собов взаимодействия основного приложения и динамической библиотеки. В рамках III уровня индивидуальной работы рекомендуется использовать явный способ подключения такой библиотеки.

Для выполнения ИР III уровня необходимо:

– разработать главную функцию ***WinMain***, в которой создается основное окно программы и запускается цикл обработки сообщений;

– разработать оконную процедуру, обеспечивающую обработку сообщений для основного окна программы;

– разработать функцию, которая запускается в рамках созданного потока в процедуре обработки сообщений (при обработке сообщения ***WM_CREATE***), загружает динамическую библиотеку и выполняет вызов функции из динамической библиотеки; все три упомянутые выше функции размещаются в одном файле;

– создать в отдельном файле динамическую библиотеку, содержащую единственную функцию, которая реализует требования выбранного варианта задания.

Пример создания программы динамической библиотеки приведен в листинге 7.1.

Листинг 7.1

```
1: #include <stdio.h>
2: extern "C" __declspec(dllexport) int Information(char *InfoString)
3: {
4:         sprintf(InfoString," Управление ресурсами");
5:         return 0;
6: }
```

В листинге 7.1 предполагается, что в единственную функцию DLL с именем ***Information*** передается параметр – строка ***InfoString***. При выполнении ИР в функции динамической библиотеки должны быть реализованы требования выполняемого варианта задания.

Важным моментом является описание функции. Функция должна быть описана как внешняя, реализованная на языке Си и специфицирована как экспортируемая функция динамической библиотеки.

В тексте программы основного приложения (при выполнении ИР III уровня в функции, которая запускается в рамках созданного потока в процедуре обработки сообщений вызов функции из динамической библиотеки может быть осуществлен способом, который приведен ниже в листинге 7.2 (явный способ подключения библиотеки).

Листинг 7.2

```
1:  typedef int (*ImportFunction)(char *);
2:  char String[256];
3:  ImportFunction DLLInfo;
4:  HINSTANCE hinstLib = LoadLibrary(TEXT("info.dll"));
5:  DLLInfo = (ImportFunction)GetProcAddress(hinstLib, "Information");
6:  DLLInfo(String);
7:  FreeLibrary(hinstLib);
```

В 1 строке листинга 7.2 определен тип указателя на функцию. Во 2 строке листинга объявлен символьный массив, который будет передан в функцию динамической библиотеки. Затем в строке 3 объявлена переменная – указатель на функцию. В строке 4 происходит загрузка динамической библиотеки ***info.dll***, текст которой был приведен в листинге 7.1. Затем полученный в строке 4 дескриптор динамической библиотеки используется для загрузки функции ***Information***, определенной в динамической библиотеке (строка 5). После этого в строке 6 выполняется вызов функции из динамической библиотеки. При вызове функции передается параметр ***String*** – символьный массив. В строке 7 динамическая библиотека закрывается путем освобождения дескриптора.

8. ВЫПОЛНЕНИЕ ИР IV УРОВНЯ. ИСПОЛЬЗОВАНИЕ ЯЗЫКА АССЕМБЛЕРА

Для разработки наиболее критичных с точки зрения производительности фрагментов кода программного обеспечения, а также для работы с процессором или другой аппаратной частью, иногда прибегают к ассемблерным вставкам в язык высокого уровня.

Целью, стоящей перед студентами при выполнении ИР IV уровня, является освоение технологии использования языка Ассемблера при разработке программного обеспечения в рамках ОС Windows.

При выполнении IV уровня индивидуальной работы в функции из динамической библиотеки дополнительно вызывается ассемблерная вставка, выполняющая дополнительные требования, содержащиеся в выбранном варианте задания (варианты заданий см. в табл. 3.1). Полу-

ченная информация из ассемблерной вставки выводится вместе с результатом решения задачи в рамках I, II, III уровня. В случае отсутствия исследуемого компонента архитектуры процессора программа должна сообщать об этом (например, отсутствие КЭШа III уровня).

Следует отметить, что выполнение требований IV уровня имеет повышенный уровень сложности и предполагает самостоятельный поиск и изучение студентом программных средств, позволяющих реализовать указанное задание.

Ниже приведен пример использования ассемблерной вставки для обращения к инструкции процессора ***cpuid*** с целью получения сведений о производителе процессора.

Листинг 8.1

```
1:  void mycpuid(int regs[4], int func)
2:  {
3:      int ieax, iebx, iecx, iedx;
4:      __asm
5:      {
6:              mov eax,func
7:              cpuid
8:              mov ieax,eax
9:              mov iebx,ebx
10:             mov iecx,ecx
11:             mov iedx,edx
12:     }
13:     regs[0] = ieax;
14:     regs[1] = iebx;
15:     regs[2] = iecx;
16:     regs[3] = iedx;
17:     }
```

В функцию ***mycpuid*** передается массив из 4 целочисленных значений и номер функции, которую необходимо вызвать при обращении к инструкции процессора ***cpuid***. Для получения сведений о производителе процессора используются номера 0x80000002, 0x80000003, 0x80000004. Подробно о параметрах и значениях команды ***cpuid*** в процессорах Intel и AMD можно найти в документации [5, 6].

В строке 6 происходит установка функции команды ***cpuid***, и в строке 7 эта инструкция вызывается. В строках 8–11 полученные в ре-

гистрах процессора результаты выполнения команды ***cpuid*** копируются и далее в строках 13–16 записываются в переданный в функцию массив значений. Ниже приведен листинг функции-оболочки, вызывающей функцию ***mycpuid***.

Листинг 8.2

```
1:   int CPUInfo[4];
2:   char CPUBrandString[64];
3:   mycpuid(CPUInfo, 0x80000002);
4:   memcpy(CPUBrandString, CPUInfo, sizeof(CPUInfo));
5:   mycpuid(CPUInfo, 0x80000003);
6:   memcpy(CPUBrandString+16, CPUInfo, sizeof(CPUInfo));
7:   mycpuid(CPUInfo, 0x80000004);
8:   memcpy(CPUBrandString+32, CPUInfo, sizeof(CPUInfo));
```

После каждого обращения к функции ***mycpuid*** результаты возвращаются в массиве ***CPUInfo*** (строки 3, 5, 7), а затем копируются в массив названия процессора (строки 4, 6, 8). В результате массив ***CPUBrandString*** будет содержать название процессора.

Следует также иметь в виду, что в различных процессорах номер команды ***cpuid*** для чтения необходимой информации различен. При выполнении индивидуальной работы этот факт должен быть учтен. Приветствуется, по возможности, тестирование программы на процессорах различных фирм-изготовителей.

9. ПОРЯДОК ВЫПОЛНЕНИЯ И СДАЧИ/ЗАЩИТЫ ИНДИВИДУАЛЬНОЙ РАБОТЫ

1. Разработка программного приложения по теме «Управление ресурсами в Windows» выполняется каждым студентом индивидуально.

2. Студент выбирает вариант задания, а также тот уровень, на котором предполагается выполнение задания.

Выбранный номер варианта сообщается преподавателю, ведущему лабораторные занятия по курсу в группе. Для студентов одной группы выбранные номера вариантов не должны повторяться.

3. Студент самостоятельно (в терминальном классе факультета или дома) выполняет индивидуальную работу в соответствии с выбранным

вариантом и уровнем сложности. За консультацией по вопросам выполнения индивидуальной работы студенты обращаются к преподавателям, ведущим лабораторные занятия по курсу в группе, во время лабораторных работ или в другое назначенное преподавателем время.

4. Для проверки выполненной индивидуальной работы студент по электронной почте пересылает преподавателю, осуществляющему проверку выполнения индивидуальной работы (преподавателю, ведущему лабораторные занятия по курсу в группе), электронное письмо, содержащее:

– ФИО студента, номер группы, номер варианта задания и выбранный уровень индивидуальной работы, а также свой e-mail;

– комплект файлов с результатами выполнения индивидуальной работы.

5. Комплект файлов с результатами выполнения индивидуальной работы включает:

1) файл с текстами программ основного приложения:

– I уровень: консольное приложение, создающее вторичный поток и функция, запускаемая в рамках созданного потока и реализующая требования выбранного варианта задания;

– II уровень: главная функция ***WinMain***, процедура обработки сообщений для основного окна программы, функция, запускаемая в рамках созданного потока в процедуре обработки сообщений и реализующая требования выбранного варианта задания;

– III, IV уровень: главная функция ***WinMain***, процедура обработки сообщений для основного окна программы, функция, запускаемая в рамках созданного потока в процедуре обработки сообщений, загружающая динамическую библиотеку и вызывающая функции из динамической библиотеки.

2) файл с текстами программ динамической библиотеки для III, IV уровня:

– III уровень: функция, реализующая требования выбранного варианта задания;

– IV уровень: функция, реализующая требования выбранного варианта задания и ассемблерная вставка.

3) файл сценария компиляции основного приложения и динамической библиотеки (compile.bat);

4) файл с отчетом о проделанной работе (doc.-файл с описанием разработанных программ, логики их работы, используемых программных средств и пр.).

Все разработанные программы должны быть документированы.

Все файлы должны быть запакованы в ZIP-архив.

6. Файл сценария компиляции основного приложения и динамической библиотеки (compile.bat) определяет процесс их подготовки.

Ниже приведен пример листинга файла сценария для компиляции основного приложения и динамической библиотеки в предположении, что основное приложение содержится в файле ***driver.cpp***, а динамическая библиотека – в файле ***info.cpp***.

Листинг 9.1

```
1:  if exist *.exe del *.exe
2:  if exist *.dll del *.dll
3:  cl driver.cpp kernel32.lib user32.lib gdi32.lib advapi32.lib
4:  cl /LD info.cpp kernel32.lib user32.lib gdi32.lib advapi32.lib
5:  del *.obj *.lib *.exp
```

В 1 строке листинга 9.1 проверяются и удаляются уже созданные выполняемые файлы. Во 2 строке то же самое делается с динамическими библиотеками. В строках 3 и 4 компилируются файлы приложения и библиотеки. Ключ ***/LD*** задает инструкцию создания библиотеки (динамической, статической и карты библиотеки, которые удаляются в силу ненадобности в строке 5). Файлы ****.lib*** указанные в строках 3 и 4 необходимы для подключения функций Windows API.

7. Проверка преподавателем результатов выполнения индивидуальной работы включает:

– разархивирование представленных файлов;

– анализ представленного студентом отчета о проделанной работе;

– запуск файла компиляции из командной строки с установленными переменными среды Visual Studio 2008 (это консольное приложение можно запустить ***Пуск > Все программы > Microsoft Visual Studio 2008 > Visual Studio Tools > Командная строка Visual Studio 2008*** в ОС Windows 7 и аналогично в других системах Windows);

– запуск и проверка результатов работы приложения.

8. О результатах проверки индивидуальной работы преподаватель извещает студента по электронной почте, либо назначает время для очной защиты.

9. Преподаватель, осуществляющий прием индивидуальной работы, может вносить изменения в общий порядок выполнения и сдачи/защиты индивидуальной работы.

ПРИЛОЖЕНИЕ

В приложении приведем полный текст основной программы (листинг 10.1) и динамической библиотеки (листинг 10.2) для задачи определения доступного дискового пространства в четвертом варианте сложности. Ассемблерная вставка используется для получения сведений о производителе процессора.

Листинг 10.1

```
#include <windows.h>
#include <stdio.h>
#include <string.h>
typedef int (*ImportFunction)(char *);
LRESULT CALLBACK WindowFunc(   HWND, UINT,
                               WPARAM, LPARAM);
LPCSTR         szClassName = "ControlSource";
LPCSTR         szTitle     = "Курсовик по управлению ресурсами";
char Info[256];
DWORD WINAPI ThreadFunc(void *)
{
   ImportFunction DLLInfo;
   HINSTANCE hinstLib = LoadLibrary(TEXT("info.dll"));
   DLLInfo = (ImportFunction)GetProcAddress(hinstLib, "Information");
   DLLInfo(Info);
   FreeLibrary(hinstLib);
   return 0;
}

LRESULT CALLBACK WindowFunc(HWND hWnd,
                UINT msg, WPARAM wParam, LPARAM lParam)
{
   HANDLE  hThread;
   DWORD   IDThread;
   PAINTSTRUCT ps;
   HDC hDC;
   switch(msg)
   {
```

```
    case WM_CREATE:
        hThread = CreateThread(NULL, 0, ThreadFunc,
                                    NULL, 0, &IDThread);
        WaitForSingleObject(hThread, INFINITE);
        CloseHandle(hThread);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    case WM_PAINT:
        hDC = BeginPaint(hWnd, &ps);
        TextOut (hDC, 10, 10, Info, strlen(Info));
        EndPaint(hWnd, &ps);
        break;
    default:
         return DefWindowProc(hWnd,msg,wParam,lParam);
   }
   return 0;
}

int WINAPI WinMain(HINSTANCE hThisInst,
                   HINSTANCE hPrevInst, LPSTR str, int nWinMode)
{
   MSG msg;
   WNDCLASS wcl;
   HWND hWnd;

   wcl.hInstance = hThisInst;
   wcl.lpszClassName = szClassName;
   wcl.lpfnWndProc = WindowFunc;
   wcl.style = CS_HREDRAW | CS_VREDRAW;
   wcl.hIcon   = LoadIcon(NULL,IDI_APPLICATION);
   wcl.hCursor = LoadCursor(NULL,IDC_ARROW);
   wcl.lpszMenuName = NULL;
   wcl.cbClsExtra = 0;
   wcl.cbWndExtra = 0;
   wcl.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
   RegisterClass(&wcl);
   hWnd = CreateWindow(szClassName, szTitle,
```

```
                    WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN |
                    WS_CLIPSIBLINGS,
                  100, 50, 700, 120, HWND_DESKTOP,
                    NULL, hThisInst, NULL);
   ShowWindow(hWnd,nWinMode);
   UpdateWindow(hWnd);
   while(GetMessage(&msg,NULL,0,0))
   {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
   }
   return msg.wParam;
}
```

Листинг 10.2

```
#include <windows.h>
#include <stdio.h>
void mycpuid(int regs[4], int func)
{
  int ieax, iebx, iecx, iedx;
  __asm
  {
    mov eax,func
    cpuid
    mov ieax,eax
    mov iebx,ebx
    mov iecx,ecx
    mov iedx,edx
  }
  regs[0] = ieax;
  regs[1] = iebx;
  regs[2] = iecx;
  regs[3] = iedx;
}

extern "C" __declspec(dllexport) int Information(char *InfoString)
{
    DWORD     dwSectPerClust, dwBytesPerSect,
              dwFreeClusters, dwTotalClusters;
    unsigned __int64 i64TotalBytes, i64FreeBytes;
```

```
    float f64TotalBytes, f64FreeBytes;
    int CPUInfo[4];
    char CPUBrandString[64];
    mycpuid(CPUInfo, 0x80000002);
    memcpy(CPUBrandString, CPUInfo, sizeof(CPUInfo));
    mycpuid(CPUInfo, 0x80000003);
    memcpy(CPUBrandString+16, CPUInfo, sizeof(CPUInfo));
    mycpuid(CPUInfo, 0x80000004);
    memcpy(CPUBrandString+32, CPUInfo, sizeof(CPUInfo));
    GetDiskFreeSpace(TEXT("C:\\"),&dwSectPerClust,&dwBytesPerSect,
                     &dwFreeClusters,&dwTotalClusters);
    i64TotalBytes = (__int64)dwTotalClusters * dwSectPerClust *
dwBytesPerSect;
    i64FreeBytes  = (__int64)dwFreeClusters * dwSectPerClust *
dwBytesPerSect;
    f64TotalBytes = i64TotalBytes /(1024.0f*1024*1024);
    f64FreeBytes  = i64FreeBytes / (1024.0f*1024*1024);
    sprintf(InfoString,"Disk C: free %.2f GB from %.2f GB,  CPU brand:
%s \n",
          f64FreeBytes,f64TotalBytes,CPUBrandString);
    return 0;
}
```

Результатом работы является окно, в которое приложение выводит информацию о доступном дисковом пространстве и о производителе процессора.

Рис. 1

ЛИТЕРАТУРА

1. *Побегайло А.* Системное программирование в Windows. – СПб.: – Петербург, 2006. – 1056 с.
2. *Рихтер Дж.* Windows для профессионалов. Изд-во: Питер, 2001. – 722 с.
3. *Щупак Ю.* Win32 API. Эффективная разработка приложений. – СПб.: – Питер, 2007. – 572 с.
4. Visual C++ и MFC. / А. Мешков, Ю. Тихомиров. – М.: Изд-во БХВ, 1999. – 1020 с.
5. http://www.intel.com/Assets/PDF/appnote/241618.pdf.
6. http://support.amd.com/us/Embedded_TechDocs/25481.pdf.
7. *Тракимус Ю.В.* Разработка простых консольных приложений с помощью Microsoft Visual Studio 2008. Изд-во: НГТУ, 2010. – 44 с.
8. Современные операционные системы / С.В. Назаров, А.И. Широков. – М.: Интернет-Университет Информационных технологий: БИНОМ. Лаборатория знаний, 2011. – 279 с.

СОДЕРЖАНИЕ

УПРАВЛЕНИЕ РЕСУРСАМИ
В ОС WINDOWS

Методические указания

Редактор *М.Е. Мельникова*
Выпускающий редактор *И.П. Брованова*
Компьютерная верстка *В.Н. Зенина*

Подписано в печать 25.05.2012. Формат 60×84 1/16. Бумага офсетная. Тираж 100 экз.
Уч.-изд. л. 1,86. Печ. л. 2,0. Изд. № 57. Заказ № Цена договорная

Отпечатано в типографии
Новосибирского государственного технического университета
630092, г. Новосибирск, пр. К. Маркса, 20