Министерство образования и науки Российской Федерации

ФГБОУ ВО «Дагестанский государственный технический университет»

Учебно-методические указания

по выполнению лабораторных работ

по дисциплине «Системное программное обеспечение»

для бакалавров направления подготовки

09.03.01 – Информатика и вычислительная техника

Махачкала 2019

УДК 681.31(031)

Учебно-методические указания по выполнению лабораторных работ по дисциплине «Системное программное обеспечение» для бакалавров направления подготовки 09.03.01 – Информатика и вычислительная техника / Сост.: У.А.Мусаева; ДГТУ. Каф. УиИТСиВТ. – Махачкала.: Изд. ДГТУ, 2019 – 42 с.

Содержат описания лабораторных работ №№1, 2, указания к их выполнению, задания и требования к оформлению отчета. Изложены основы разработки программ системного назначения для операционных систем семейства Windows. Рассмотрены вопросы работы с интерфейсом прикладного программирования WinAPI. Приведены исходные тексты примеров программ и рекомендуемая литература.

Табл. 6. Библиогр.: 8 назв.

Составитель: к.т.н., доцент кафедры УИТСиВТ Мусаева У.А.

Рецензенты: зав. каф. УиИТСиВТ, д.т.н., профессор Саркаров Т.Э.,
ЦГА РД,нач. отдела Автоматизированных АТ Мусаев Г.М.

Печатается по постановлению ученого Совета Дагестанского государственного технического университета от_____ 20__ г.

Содержание

Введение

Целью лабораторных работы является закрепление основ и углубление знаний в области устройства современных операционных систем семейства Windows с точки зрения программиста и получения практического опыта написания программ системного назначения, а так же ознакомление студентов со средствами компиляции и отладки программ, которые предназначены для низкоуровневого взаимодействия с ОС.

При выполнении лабораторных работ должен соблюдаться следующий порядок выполнения работы:

- ознакомиться с описанием лабораторной работы;
- получить номер варианта задания у преподавателя;
- изучить необходимый теоретический материал, пользуясь настоящими указаниями и рекомендованной литературой;
- написать программу и отладить ее;
- подготовиться к ответам на теоретические вопросы по теме лабораторной работы;
- оформить отчет.

Все студенты должны предъявить индивидуальный отчет о результатах выполнения лабораторной работы. Отчет должен содержать следующие пункты:

1) Титульный лист.

2) Краткое теоретическое описание.

3) Задание на лабораторную работу, включающее четкую формулировку задачи.

4) Листинг программы.

5) Результаты выполнения работы.

При сдаче отчета студент должен показать знание теоретического материала в объеме, определяемом тематикой лабораторной работы, а также пониманием сущности выполняемой работы.

Лабораторная работа № 1. Структура программы для Windows на языке C++

1. Цель работы

Цель работы – изучить структуру программы для операционной системы Windows. Научится создавать и регистрировать окно на основе класса окна. Изучить структуру и состав оконной процедуры.

2. Краткие теоретические сведения

2.1 Первая программа для Windows

Расммотрим пример простейшей программы для Windows.

```
//HELLOWIN.C
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR
szCmdLine, intiCmdShow)
{
   static char szAppName[] = "HelloWin";
   HWND hwnd;
   MSG  msg;

   WNDCLASSEX wndclass;
   wndclass.cbSize        = sizeof(wndclass);
   wndclass.style         = CS_HREDRAW | CS_VREDRAW;
   wndclass.lpfnWndProc   = WndProc;
   wndclass.cbClsExtra    = 0;
   wndclass.cbWndExtra    = 0;
   wndclass.hInstance     = hInstance;
   wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
   wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
   wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
   wndclass.lpszMenuName  = NULL;
   wndclass.lpszClassName = szAppName;
   wndclass.hIconSm       = LoadIcon (NULL, IDI_APPLICATION);

   RegisterClassEx (&wndclass);
   hwnd = CreateWindow (szAppName,  // имя класса окна
      "The Hello Program",          // заголовок окна
      WS_OVERLAPPEDWINDOW,          // стиль окна
      CW_USEDEFAULT,                // начальная позиция X
      CW_USEDEFAULT,                // начальная позиция y
      CW_USEDEFAULT,                // ширина окна
      CW_USEDEFAULT,                // высота окна
      NULL,                         //описатель родительского окна
      NULL,                         // описатель меню окна
      hInstance,                    // описатель экземпляра программы
      NULL) ;                       // параметры создания
   ShowWindow (hwnd, iCmdShow);
   UpdateWindow (hwnd);
```

```
    while (GetMessage (&msg, NULL, 0, 0))
    {
       TranslateMessage (&msg);
       DispatchMessage (&msg);
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM
lParam)
{
    HDC         hdc;
    PAINTSTRUCT ps;
    RECT        rect;

    switch (iMsg)
    {
       case WM_PAINT:
          hdc = BeginPaint (hwnd, &ps);
          GetClientRect (hwnd, &rect);
          DrawText (hdc, "Hello, Windows!", -1, &rect, DT_SINGLELINE |
DT_CENTER | DT_VCENTER);
          EndPaint (hwnd, &ps);
          return 0;
       case WM_DESTROY:
          PostQuitMessage (0);
          return 0;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

Программа создает обычное окно приложения. В окне, в центре рабочей области, выводится текст “Hello, Windows!”. При изменении размеров окна программа будет автоматически перемещать строку текста “Hello,Windows!” в новый центр рабочей области окна.

2.2 Настройки проекта

Windows поддерживает работу как с Юникод-строками, так и с однобайтными кодировками. Для этого в ее состав включены два набора функций: например существует две версии DrawTextW (Unicode) и DrawTextA (ANSI), а сама используемая в программе DrawText, по сути, является заглушкой для вызова одной из указанных выше функций.

При вызове функций WinAPI, в зависимости от настроек среды разработки и установленных флагов, вызывается либо Юникод-версия функции, либо обычная, однобайтная ANSI.

По-умолчанию современные версии сред разработки работают с Юникод-версиями функций, однако для простоты работы, далее в примерах подразумевается, что работа ведется с однобайтными кодировками.

Чтобы в настройках проекта указать какой тип строк используется по умолчанию необходимо во вкладке «General» выбрать для параметра «Character Set» значение «Use Multi-Byte Character Set» Вместо «Use Unicode character Set» (рисунок 1.1).

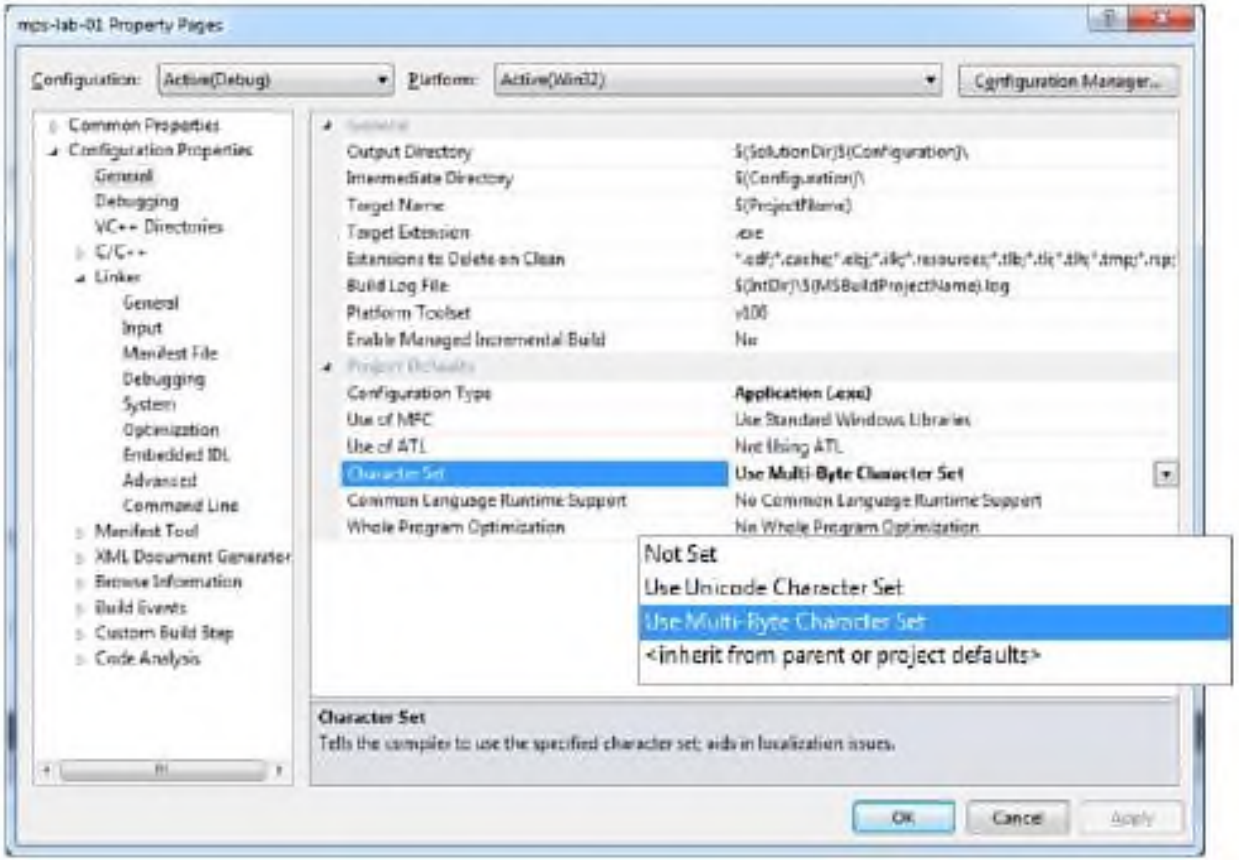

Рисунок 1.1 – Настройки среды разработки

2.3 Файл исходного текста программы на языке С

В файле имеется только две функции: WinMain и WndProc. WinMain – это точка входа в программу. WndProc –это “оконная процедура”. Каждое окно, не

зависимо от того каким оно является, имеет свою оконную процедуру. Оконная процедура – это способ инкапсулирования кода, отвечающего за ввод информации (обычно с клавиатуры или мыши) и за вывод информации на экран. WndProc вызывается только из Windows. Однако в WinMain имеется ссылка на WndProc, поэтому эта функция описывается в самом начале программы, еще до определения WinMain.

2.3.1 Используемые функции Windows.

HELLOWIN.C использует не менее 16-ти функций Windows. Здесь перечислены эти функции в порядке их появления в программе.

LoadIcon – загружает значок для использования в программе.

LoadCursor – загружает курсор мыши для использования в программе.

GetStockObject – получает графический объект (в данном случае кисть).

RegistrClassEx – регистрирует класс окна для определенного окна программы.

CreateWindow – создает окно на основе класса окна.

ShowWindow – выводит окно на экран.

UpdateWindow – заставляет окно перерисовать своё содержимое.

GetMessage – получает сообщение из очереди сообщений.

TranslateMessage – преобразует некоторые сообщения, полученные с помощью клавиатуры.

DispatchMessage – отправляет сообщение оконной процедуре.

BeginPaint – инициализирует начало процесса рисования окна.

GetClientRect – получает размер рабочей области окна.

DrawText – выводит текст.

EndPaint – прекращает рисование окна.

PostQuitMessage – вставляет сообщение “завершить” в очередь сообщений.

DefWindowProc – выполняет обработку сообщений по умолчанию.

2.3.2 Идентификаторы

Эти идентификаторы записаны в заголовочных файлах Windows. Некоторые из этих идентификаторов содержат двух или трёхбуквенный префикс, за которым следует символ подчеркивания: CS_HREDRAW, CS_VREDRAW, CW_USEDEFAULT, DT_CENTER, DT_SINGLELINE, DT_VCENTER, IDC_ARROW, IDI_APPLICATION, SND_ASYNC, SND_FILENAME, WM_DESTROY, WM_PAINT – это просто числовые константы. Префикс показывает основную категорию, к которой принадлежит константа, как показано в таблице 1.1.

Таблица 1.1 – Префиксы категорий идентификаторов

Префикс	Категория
CS	Опция стиля класса
IDI	Идентификационный номер иконки
IDC	Идентификационный номер курсора
WS	Стиль окна
CW	Опция создания окна
WM	Сообщение окна
SND	Опция звука
DT	Опция рисования текста

2.3.3 Типы данных в приложениях Windows

UINT – Беззнаковое целое.

PSTR – Указатель на строку, т.е. char*.

WPARAM – Беззнаковое короткое целое.

LPARAM – 32-разрядное знаковое длинное целое.

Функция WndProc возвращает значение типа LRESULT. Оно определяется просто как LONG. Функция WinMain получает тип WINAPI, а

функция WndProc получает тип CALLBACK. Оба эти идентификатора определяются как stdcall, что является ссылкой на особую последовательность вызовов функций, которая имеет место между самой ОС и ее приложением.

В HELLOWIN используется 4 структуры данных, определенных в заголовочных файлах Windows. Этими структурами являются:

Таблица 1.2 – Структуры WinAPI

Структура	Значение
MSG	Структура сообщения.
WNDCLASSEX	Структура класса окна.
PAINTSTRUCT	Структура рисования.
RECT	Структура прямоугольника.

Первые две структуры данных используются в WinMain для определения структур, названных msg и wndclass. Две вторые используются в WndProc для определения структур ps и rect. Имеется еще идентификаторы написаных прописными буквами. Описатель – это просто число, которое ссылается на объект (обычно длиной 32-бита).

Таблица 1.3 – Структуры WinAPI

Идентификатор	Значение
HINSTANCE	Описатель экземпляра программы.
HWND	Описатель окна.
HDC	Описатель контекста устройства.
HICON	Описатель значка.
HCURSOR	Описатель курсора.
HBRUSH	Описатель кисти.

2.4 Точка входа программы

Текст программы начинается с инструкции #include, которая позволяет включить в программу заголовочный файл WINDOWS.H. WINDOWS.H включает в себя много других заголовочных файлов, содержащих объявление функций, структур, новые типы данных и константы Windows. Затем следует объявление WndProc:

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

Это объявление необходимо потому, что внутри WinMain имеются ссылки на WndProc. Точкой входа программы для Windows является функция WinMain. WinMain всегда определяется следующим образом:

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR szCmdLine, int iCmdShow).

Эта функций использует последовательность вызовов WINAPI и, по своему завершению, возвращает ОС целое. В ней есть 4 параметра:

- Параметр hInstance называется описателем экземпляра. Это уникальное число, идентифицирующее программу, когда она работает под Windows.

- Параметр hPrewInstance – предыдущий экземпляр – в настоящее время устарел.

- Параметр szCmdLine – это указатель на строку оканчивающуюся нулем, в которой содержатся любые параметры, переданные программе из командной строки.

- Параметр iCmdShow – число, показывающее, каким должно быть выведено окно на экран в начальный момент. Это число задаётся при запуске программы другой программой.

Таблица 1.4 – Варианты значений параметра iCmdShow

Идентификатор	Назначение
SW_SHOWNORMAL	Вывести обычное окно
SW_HIDE	Скрывает окно и активизирует другое окно
SW_MINIMIZE	Минимизирует определенное окно и активизирует окно верхнего уровня в списке системы.
SW_MAXIMIZE	Максимизирует определенное окно.
SW_RESTORE	Активизирует и отображает окно. Если окно минимизировано или максимизировано, Windows восстанавливает его до первоначального размера и позиции (так же как SW_SHOWNORMAL).
SW_SHOW	Активизирует окно и отображает его в текущем размере и позиции.

Таблица 1.4 (Продолжение)

SW_SHOWMINIMIZE	Активизирует окно и отображает его как минимизированное окно
SW_SHOWMAXIMIZE	Активизирует окно и отображает его как максимизируемое окно
SW_SHOWDEFAULT	Устанавливает состояние, основанное на флажке SW_, определенном в структуре STARTUPINFO, переданной к функции CreateProcess программой, которая начала приложение.
SW_SHOWNA	Отображает окно не активизируя его.
SW_SHOWNOACTIVE	Отображает окно в самом последнем размере и позиции, не активизируя его.
SW_SHOWMINNOACTIVE	Вывести свернутое окно, не активизируя его.

2.5 Регистрация класса окна

Окно всегда создаётся на основе класса окна. Класс окна идентифицирует оконную процедуру, которая выполняет процесс обработки сообщений, поступающих окну. На основе одного класса окна можно создать несколько окон. Перед созданием окна для вашей программы необходимо зарегистрировать класс окна путем вызова функции RegistrClassEx. Это расширенная версия функции RegistrClass из предыдущей версии Windows. У функции RegistrClassEx имеется один параметр: указатель на структуру типа WNDCLASSEX. Структура WNDCLASSEX определяется в заголовочных файлах Windows следующим образом:

```
typedef struct tagWNDCLASSEX
{
    UINT cbSize;
    UINT style;
    WNDPROC lpfnWndProc;
    int cbClsExtra;
    int cbWndExtra;
    HINSTANCE hInstance;
    HICON hIcon;
    HCURSOR hCursor;
    HBRUSH hbrBackground;
    LPCSTR lpszMenuName;
    LPCSTR lpszClassName;
    HICON hIconSm;
}
WNDCLASSEX;
```

В WinMain необходимо определить структуру типа WNDCLASSEX, обычно это делается так:

```
WNDCLASSEX wndclass;
```

Затем задаются 12 полей структуры и вызывается RegisterClassEx:

```
RegisterClassEx(&wndclass);
```

Наиболее важными являются второе и третье от конца поля. Второе от конца поле является именем класса окна. Третье поле является адресом оконной процедуры, которая используется для всех окон, созданных на основе данного класса. Другие поля описывают характеристики всех окон, создаваемых на основе этого класса окна. Поле cbSize равно длине структуры. Инструкция:

```
wndclass.style = CS_HREDRAW | CS_VREDRAW;
```

осуществляет объединение двух идентификаторов стиля класса с помощью поразрядной операции OR. Эти два идентификатора показывают, что все окна, созданные на основе данного класса должны целиком перерисовываться при вертикальном или горизонтальном изменении размеров окна (таблица 1.5).

Третье поле структуры WNDCLASSEX инициализируется с помощью инструкции: wndclass.lpfnWndProc = WndProc;

Эта инструкция устанавливает оконную WndProc как оконную процедуру данного окна. Она будет обрабатывать все сообщения всем окнам, созданным на основе данного класса окна. Следующие две инструкции:

```
wndclass.cbClsExtra = 0;
wndclass.cbWndExtra = 0;
```

резервируют некоторое дополнительное пространство в структуре класса и структуре окна.

Программа может использовать это пространство по своему усмотрению. В следующем поле находится описатель экземпляра класса:

```
wndclass.hInstance = hInstance;
```

Инструкции:

```
wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
```

wndclass.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

устанавливают значок для всех окон созданных на основе данного класса окна.

Таблица 1.5 – Таблица стилей класса

Стиль	Описание
CS_BYTEALIGNCLIENT	Выравнивание клиентской области окна на границе байта (по X) для расширения эффективность при выполнении операторов рисования.
CS_BYTEALIGNWINDOW	Выравнивает окно на границе байта (по X). Расширяет эффективность действий при изменении размеров и положения окна
CS_CLASSDC	Распределяет один контекст устройства между всеми окнами в классе.
CS_DBLCLKS	Посылает сообщение двойного щелчка оконной процедуре в том случае если двойной щелчок произошел в клиентской части
CS_GLOBALCLASS	Допускает, чтобы приложение создало окно класса независимо от значения hInstance, переданного функцией CreateWindow или CreateWindowEx
CS_HREDRAW	Перерисовка окна при изменении его ширины
CS_NOCLOSE	Отключает команду Close на Системном меню
CS_OWNDC	Распределяет уникальный контекст устройства для каждого окна в классе
CS_PARENTDC	Устанавливает область отсечения дочернего окна в родительском окне так, чтобы дочернее окно могло быть внутри родительского. Окно с CS_PARENTDC стилем получает регулярный контекст устройства от кэша системы контекстов устройства. Расширяет эффективность приложения.
CS_SAVEBITS	Сохраняет, как растр, блок отображаемого изображения, затененного окном. . Windows использует сохраненный растр, чтобы вновь создать отображаемое изображение, когда окно удалено. Windows отображает растр в первоначальном расположении и не посылает сообщения WM_PAINT окнам, затененным текущим окном, если память, используемая растром не была сброшена и если другие экранные действия не объявили неверным сохраненное изображение.
CS_VREDRAW	Перерисовка окна при изменении его высоты

Инструкция:

wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);

устанавливает курсор для всех окон, созданных на основе данного класса окна.

Инструкция:

wndclass.hbrBackground = GetStockObject(WHITE_BRUSH);

устанавливает цвет фона окна.

LLOWIN меню отсутствует, поэтому поле установлено в NULL:

wndclass.lpszMenuName = NULL;

На последнем этапе классу присваивается имя:

wndclass.lpszClassName = szAppName;

После описания 12 полей вызывается функция RegisterClassEx для регистрации класса:

RegisterClassEx(&wndclass);

2.6 Создание окна

Класс окна определяет основные характеристики окна, что позволяет использовать один и тот же класс для создания множества различных окон. Вызов функции CreateWindow, фактически создает окно, таким образом, детализируется информация об окне. Вызов функции CreateWindow требует передачи информации об окне в качестве параметров. В HELLOWIN.C это выглядит так:

```
hwnd = CreateWindow (
szAppName, // имя класса окна
“The Hello Program”, // заголовок окна
WS_OVERLAPPEDWINDOW, // стиль окна
CW_USERDEFAULT, // начальное положение по x
CW_USERDEFAULT, // начальное положение по y
CW_USERDEFAULT, // начальный размер по x
CW_USERDEFAULT, // начальный размер по y
NULL, // описатель родительского окна
NULL, // описатель меню
hInstance, // описатель экземпляра класса
NULL); // параметры создания
```

Параметр с комментарием “имя класса окна” – szAppName содержит строку “HelllоWin”, являющуюся именем только что зарегистрированного класса. Окно, созданное программой, является обычным перекрывающимся окном с заголовком, системным меню слева, иконками для сворачивания, закрытия и развертывания на весь экран. Это стандартный стиль окон, он называется WS_OVERLAPPEDWINDOW и помечен комментарием “стиль окна”. Комментарием “заголовок окна ” помечен текст выводимый в строке заголовка. Затем идут 4 параметра размеров и положения окна (идентификатор CW_USERDEFAULT позволяет установить все данные по умолчанию). Описатель родительского окна равен NULL т.к. у этого окна нет родительского. Описатель меню также равен NULL потому, что его нет. И наконец, параметр создания равен NULL. При необходимости этот параметр используется в качестве указателя на какое ни будь данное. Вызов CreateWindow возвращает описатель созданного класса. Этот описатель хранится в переменной hwnd, которая имеет тип HWND. У каждого окна в Windows имеется свой описатель. В нашей программе описатель используется для того, чтобы ссылаться на окно.

2.7 Отображение окна

Для отображения окна нужно сделать еще два вызова. Первый из них:

```
ShowWindow( hwnd, iCmdShow );
```

Первым параметром является описатель окна созданный с помощью CreateWindow. Вторым параметром является величина, передаваемая функцией WinMain. Он задаёт вид окна на экране.

Второй вызов:

```
UpdateWindow (hwnd);
```

предназначен для перерисовки рабочей области.

2.8 Цикл обработки сообщений

Windows поддерживает очередь сообщений для каждой программы, работающей в данный момент в ОС. Когда происходит ввод информации,

Windows преобразует ее в сообщение, которое помещается в очередь сообщений программы. Программа извлекает сообщение из очереди сообщений, выполняя блок команд, известный как “цикл обработки сообщений”:

```
while (GetMessage (&msg, NULL, 0, 0))
{
   TranslateMessage (&msg);
   DispatchMessage (&msg);
}
return msg.wParam;
```

Переменная msg – это структура типа MSG, которая определяется в заголовочных файлах Windows следующим образом:

```
typedef struct tagMSG
{
HWND hwnd;
UINT message;
WPARAM wParam;
LPARAM lParam;
DWORD time;
POINT pt;} MSG;
```

Тип данных POINT – это тип данных другой структуры, которая определяется так:

typedef struct tagPOINT {LONG x;LONG y;} POINT;

Вызов функции GetMessage, с которой начинается цикл обработки сообщений, извлекает сообщения из очереди сообщений: GetMessage (&msg, NULL, 0, 0) Этот вызов передаёт Windows указатель на структуру msg. Второй, третий. и четвертый параметры (0 и NULL) показывают, что программа получает сообщения от всех окон созданных ею. Поля структуры msg:

hwnd – описатель окна, для которого предназначено сообщение.

message – идентификатор сообщения. Это число, которое идентифицирует сообщение. Для каждого сообщения существует свой идентификатор, который задаётся в заголовочном файле и начинается с префикса WM.

wParam – 32-разрядный параметр сообщения, смысл и значение которого зависят от самого сообщения.

lParam – другой 32-разрядный параметр сообщения, смысл и значение которого зависят от самого сообщения.

pt – координаты курсора мыши в момент помещения сообщения в очередь.

Если поле message сообщения не равно WN_QUIT, то функция GetMessage возвращает не нулевое значение. Сообщение WN_QUIT заставляет прервать цикл обработки сообщений. Инструкция:

TranslateMessage (&msg); передаёт структуру msg обратно в Windows для преобразования какого либо сообщения от клавиатуры. Инструкция:

DispatchMessage (&msg); передаёт структуру msg обратно в Windows. Windows отправляет сообщение для его обработки соответствующей оконной процедуре – таким образом ОС вызывает оконную процедуру.

2.9 Оконная процедура

В нашей программе оконной процедурой является WndProc. Ей можно дать любое имя. В программе может содержаться более одной оконной процедуры. Оконная процедура всегда связана с определенным классом окна, который регистрируется с помощью RegisterClassEx. Функция CreateWindow создаёт окно на основе определенного класса окна. На основе одного и того же класса окна можно создать несколько окон. Оконная процедура всегда определяется следующим образом:

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam,)

Первым параметром является hwnd, описатель полученного сообщения окна. Вторым параметром является 32-разрядное число, которое идентифицирует

сообщение. Два последних параметра представляют дополнительную информацию о сообщении.

2.10. Обработка сообщений

В заголовочных файлах Windows определены идентификаторы, начинающиеся с префикса WM для каждого типа сообщений. Обычно используют конструкции switch и case для определения того, какое сообщение получила оконная процедура и то, как его обрабатывать. Если оконная процедура обрабатывает сообщение, то она должна вернуть 0, иначе если сообщение не обрабатывается, то оно должно передаваться функции DefWindowProc. Значение, возвращаемое функцией DefWindowProc, должно быть возвращаемым значением оконной процедуры.

В программе обрабатываются 3 сообщения: WM_PAINT, WM_CREATE, WM_DESTROY. Обработка сообщений в программе может выглядеть следующим образом:

```
switch (iMsg)
{
case WM_CREATE:
[обработка сообщения WM_CREATE];
return 0;
case WM_PAINT:
[обработка сообщения WM_PAINT];
return 0;
case WM_DESTROY:
[обработка сообщения WM_ DESTROY];
return 0;
}
return DefWindowProc (hwnd, iMsg, wParam, lParam,);
```

В этом коде функция DefWindowProc обрабатывает остальные сообщения по умолчанию.

2.11 Сообщение WM_PAINT

Сообщение WM_PAINT функции WndProc обрабатывается вторым. Это сообщение крайне важно при программировании по Windows. Она сообщает программе, что часть или вся рабочая область недействительна и ее надо перерисовать. При первом создании окна вся рабочая область недействительна, т.к.

программа еще ни чего в ней не рисовала. Сообщение WM_PAINT заставляет программы что-то нарисовать в рабочей области.

Обработка сообщения WM_PAINT почти всегда начинается с функции BeginPaint: hdc = BeginPaint (hwnd, &ps);

и заканчивается вызовом функции EndPaint:

EndPaint (hwnd, &ps);

Второй параметр – это указатель на структуру PAINTSTRUCT. В этой структуре содержится некоторая информация, которую оконная процедура может использовать для рисования в рабочей области. При обработке вызова BeginPaint, Windows обновляет фон рабочей области с помощью кисти, заданной в поле hbrBackground структуры WNDCLASSEX, которая использовалась при регистрации класса окна. вызов BeginPaint делает всю рабочую область активной и возвращает описатель контекста устройства. Контекст устройства описывает устройство вывода информации и его драйвер. Описатель контекста устройства необходим для вывода в рабочую область текста и графики. Но с помощью этого описателя контекста устройства (полученного из BeginPaint) вне рабочей области ничего не нарисуете. Функция EndPaint освобождает описатель контекста устройства, после чего его нельзя использовать.

После того, как вызвали BeginPaint, вызывается GetClientRect:

GetClientRect (hwnd, &rect); Второй параметр – это указатель на переменную rect типа RECT. RECT – это структура прямоугольник, определенная в заголовочном файле Windows. Она имеет 4 поля типа LONG, имена полей: left, top, right, bottom. GetClientRect помещает в эти поля размеры рабочей области. Поля left и top всегда равны 0, a right и bottom равны соответственно ширине и высоте рабочей области в пикселях.

WndProc никак не использует структуру RECT, за исключением передачи указателя на нее в качестве 4 параметра функции DrawText:

DrawText (hdc, “Hello, Windows 95!”, -1, &rect, DT_SINGLELINE |

DT_CENTER | DT_VCENTER);

DrawText рисует текст. Третий параметр равный –1, определяет, что строка оканчивается нулевым символом. Последний параметр – это набор флагов, заданных в заголовочных файлах Windows. Флаги показывают, что текст выводится по центру, в одну строку, и внутри рабочей области.

Когда рабочая часть становится недействительной, WndProc получает новое сообщение WM_PAINT и снова рисует текст в центре рабочей области.

2.12 Сообщение WM_DESTROY

Это сообщение говорит о том, что Windows находится в процессе ликвидации окна в ответ на полученную от пользователя команду. Наша программа стандартно реагирует на это сообщение, вызывая:

PostQuitMessage (0);

Эта функция ставит сообщение WM_QUIT в очередь сообщений программы. Когда GetMessage получает сообщение WM_QUIT и возвращает 0, что заставляет WinMain прекратить обработку сообщений и закончить программу.

3. Контрольные вопросы

1) Какая функция является точкой входа в программу.

2) Какие параметры у функции WinMain и что они означают.

3) С помощью каких структур осуществляется регистрация класса окна (объясните назначение полей этих процедур).

4) Какая функция создаёт окно.

5) Для чего существует сообщение WM_PAINT.

6) Для чего существует сообщение WM_DESTROY.

7) Какая функция по умолчанию обрабатывает все сообщения.

4. Лабораторное задание

1) Изучить описание лабораторной работы.

2) Составить простое приложение Windows, используя все возможные стили класса окна. Изменить координаты и размеры окна.

3) Создать приложение с различными опциями показа окна, изменяя параметр iCmdShow. Изменить заголовок окна и текст в окне.

4) Отладить и протестировать полученную программу.

5) Оформить отчёт.

Лабораторная работа № 2.

Использование контекста устройства, вывод текста, использование полос прокрутки

1. Цель работы

Изучить возможности по выводу текстовой информации в окно с использованием полос прокрутки.

2. Краткие теоретические сведения

В Windows можно выводить информацию только в рабочую часть окна. Windows – это ОС, управляющая сообщениями. Windows уведомляет приложения о различных событиях путем постановки синхронных сообщений в очередь сообщений приложения или путем отправки асинхронных сообщений соответствующей оконной процедуре (синхронные сообщения – это сообщения которые становятся в очередь сообщений, а асинхронные – это передающиеся на прямую без очереди). Посылая синхронное сообщение WM_PAINT, Windows уведомляет оконную процедуру о том, что часть рабочей области необходимо обновить.

2.1 Сообщение WM_PAINT

Большинство программ под Windows вызывают функцию UpdateWindow при инициализации в WinMain, сразу перед входом обработки сообщений. ОС использует эту возможность для асинхронной отправки в оконную процедуру первого сообщения WM_PAINT. Это сообщение информирует оконную процедуру

о готовности рабочей области к рисованию. Оконная процедура получает сообщение WM_PAINT при возникновении следующих ситуаций:

- Предварительно скрытая часть окна открылась, когда пользователь передвинул окно или совершил какое-то действие, в результате которого окно опять стало видимым;

- Пользователь изменил размеры окна;

- В программе для прокрутки части рабочей области используются функции ScrollWindow и ScrollDC;

- Для генерации сообщения WM_PAINT в программе используются функции InvalidateRect или InvalidateRgn.

- Windows посылает синхронно сообщение WM_PAINT в следующих случаях:
- Удаляется окно диалога или окно сообщения, которое перекрывало часть рабочей области;
- Раскрывается горизонтальное меню и затем удаляется с экрана;
- В некоторых случаях Windows всегда сохраняет перекрываемую область:
- При перемещении курсора мыши.
- Перемещение иконки по рабочей области.

2.2 Действительные и недействительные прямоугольники

Часто полностью перерисовывать рабочую область не нужно, а лишь ее часть. Например, когда часть рабочей области закрыто окном диалога. Перерисовка требуется только для прямоугольной области, вновь открывающейся при удалении окна диалога. Эта область называется “недействительным регионом” или ”регионом обновления”. Появление недействительного региона вынуждает Windows поместить синхронное сообщение WM_PAINT в очередь сообщений программы. В Windows для каждого окна поддерживается структура информации о рисовании. В этой структуре содержатся координаты минимально возможного прямоугольника, содержащего недействительную область. Эта информация носит название недействительного прямоугольника. Если еще один регион становится

недействительным, то ОС рассчитывает новый недействительный прямоугольник, который содержит оба регион.

Оконная процедура, вызывая функцию InvalidateRect, может задать недействительный прямоугольник в своей рабочей области. Если в очереди сообщений содержится WM_PAINT, то ОС рассчитывает новый недействительный прямоугольник, иначе помещает его в очередь сообщений. При получении сообщения WM_PAINT, оконная процедура получит и координаты недействительного прямоугольника. Однако, координаты можно получить и с помощью функции GetUpdateRect, не дожидаясь сообщения

WM_PAINT.

После того как оконная процедура вызывает функцию BeginPaint при обработке сообщения WM_PAINT, вся рабочая область становится действительной. Программа, вызвав функцию ValidateRect, также может сделать действительной любую прямоугольную зону.

2.3 Контекст устройства

Описатель – это просто число, которое ОС использует для внутренней ссылки на объект. После получения описателя, его используют в различных графических функциях. Описатель контекста устройства – это паспорт окна для функций GDI (графический интерфейс устройства). Описатель даёт возможность использовать функции GDI при выводе графики. Контекст устройства фактически является структурой данных, которая внутренне поддерживается GDI. Контекст устройства связан с конкретным устройством вывода информации, таким как принтер, плоттер или дисплей. Что касается дисплея, то в данном случае контекст устройства связан с конкретным окном на экране. Когда программе необходимо начать рисовать, она должна получить описатель контекста устройства. После окончания рисования его необходимо освободить. Когда программа освобождает описатель, он становится недействительным и не может далее использоваться.

2.4 Получение описателя контекста устройства

Для получения описателя контекста устройства обычно используют два способа.

2.4.1 Получение описателя контекста устройства при помощи BeginPaint

Применяется при обработке сообщения WM_PAINT. Применяются две функции: BeginPaint и EndPaint. Для этих функций требуется описатель окна и адрес переменной типа структуры PAINTSTRUCT. Переменная этого типа определяется внутри оконной процедуры следующим образом: PAINTSTRUCT ps;

Переменная описателя контекста устройства определяется следующим образом: HDC hdc;

Тип HDC определяется как 32-разрядное беззнаковое целое. Типовой процесс обработки сообщения WM_PAINT:

```
case WM_PAINT:
hdc = BeginPaint ( hwnd, &ps ); // начало рисования
[ использование функций GDI ]
EndPaint ( hwnd, &ps ); // конец рисования
return 0;
```

Функции BeginPaint и EndPaint должны вызываться парой. Если сообщение WM_PAINT не обрабатывается, то оно передаётся в DefWindowProc. DefWindowProc обрабатывает сообщение WM_PAINT
следующим образом:

```
case WM_PAINT:
hdc = BeginPaint ( hwnd, &ps ); // начало рисования
EndPaint ( hwnd, &ps ); // конец рисования
return 0;
```

Делает неактивную область активной, но не перерисовывает её.

2.4.2 Структура информации о рисовании

Структура информации о рисовании – это PAINTSTRUCT. Она определяется так:

```
typedef struct tagPAINTSTRUCT
{
HDC hdc;
BOOL fErase;
RECT rcPaint;
BOOL fRestore;
BOOL fIncUpdate;
BYTE rgbReserved[32];
} PAINTSTRUCT;
```

Windows заполняет поля структуры, когда программа вызывает функцию BeginPaint. Программе можно использовать только 3 первых поля, остальными пользуется ОС. Поле hdc – это описатель контекста устройства. В подавляющем большинстве случаев, в поле fErase установлен флаг TRUE, означающий, что Windows обновит фон недействительного прямоугольника. Windows перерисует фон, используя кисть, заданную в поле hbrBackground структуры WNDCLASSEX, которая использована при регистрации класса окна во время инициализации WinMain. Многие программы для Windows используют белую кисть:

wndclass.hbrBackground =(HBRUSH) GetStockObject (WHITE_BRUSH);

Однако, если, вызывается функция InvalidateRect, то делается недействительным прямоугольник рабочей зоны программы, а последний параметр этой функции определяет, необходимо ли стирать фон. Если параметр равен FALSE, Windows не будет стирать фон и поле fErase также будет равно FALSE.

Поле rcPaint структуры PAINTSTRUCT определяет границы недействительного прямоугольника. Недействительный прямоугольник – это та область, которую необходимо перерисовать.

Прямоугольник rcPaint в PAINTSTRUCT – это не только недействительный прямоугольник, это также и отсекающий прямоугольник.

Это означает, что Windows рисует только внутри отсекающего прямоугольника (т. е. если недействительная зона не является прямоугольником, Windows рисует только внутри этой зоны). Когда используется описатель контекста

устройства из структуры PAINTSTRUCT, Windows не будет рисовать вне прямоугольника rcPaint.

Чтобы при обработке сообщения WM_PAINT рисовать вне прямоугольника rcPaint, необходимо сделать вызов: InvalidateRect (hWnd, NULL, TRUE);

перед вызовом BeginPaint. Это сделает недействительной всю рабочую область и обновит ее фон. Если же значение последнего параметра будет равно FALSE, то фон обновляться не будет. Все что там было, останется неизменным.

2.4.3 Получение описателя контекста устройства при помощи GetDC

Вызывайте GetDC для получения описателя контекста устройства и ReleaseDC, если он больше не нужен:

```
hdc = GetDC (hwnd) ;
[использование функций GDI] ReleaseDC (hwnd, hdc) ;
```

Также как BeginPaint и EndPain, функции GetDC и ReleaseDC следует вызывать парой. В отличии от описателя контекста устройства, полученного из структуры PAINTSTRUCT, в описателе контекста устройства,

возвращаемом функцией GetDC, определен отсекающий прямоугольник, равный всей рабочей области. Теперь можно рисовать во всей рабочей области.

2.5 Функция TextOut

Рассмотрим функцию TextOut: TextOut (hdc, x, y, psString, iLength) ;

Первый параметр – это описатель контекста устройства, являющийся возвращаемым значением функций GetDC и BeginPain. Цвет фона текста не является цветом фона, который установлен при определении класса окна. Фон в классе окна – это кисть, являющаяся шаблоном, которая может иметь, а может и не иметь чистый цвет, и которую Windows использует для закрашивания рабочей области. Поэтому в начале программы следует задавать цвет кисти, к примеру WHITE_BRUSH. Параметр psString – это указатель на символьную строку, а iLength –длина строки. TextOut не определяет конца строки по нулевому символу.

Значения x и y определяет точку начала строки. Задаваемый по умолчанию режим отображения называется MM_TEXT. В режиме отображения MM_TEXT логические единицы соответствуют физическим, которыми являются пиксели, задаваемые относительно левого верхнего угла рабочей области.

2.6 Системный шрифт

Контекст устройства также определяет шрифт, который Windows использует при выводе текста в рабочую область. По умолчанию задается системный шрифт или SYSTEM_FONT. Системный шрифт – это шрифт, который Windows использует для текста заголовка, меню и окон диалога. Системный шрифт является растровым шрифтом, т. е. все символы определены как пиксельные шаблоны. Windows имеет несколько растровых шрифтов с разными размерами (для работы с разными видеоадаптерами).

2.7 Размер символа

Размер символа можно получить с помощью вызова функции GetTextMetrics. Для функции GetTextMetrics требуется описатель контекста устройства, поскольку ее возвращаемым значением является информация о шрифте, выбранным в данное время в контексте устройства. Windows копирует различные значения метрических параметров текста в структуру типа TEXTMETRICS. Для использования функции GetTextMetrics, во-первых, необходимо определить переменную:

TEXTMETRICS tm; Далее нужно получить описатель контекста устройства и вызвать GetTextMetrics:

```
hdc = GetDC (hwnd) ;
GetTextMetrics (hdc, &tm) ;
ReleaseDC(hwnd, hdc);
```

2.8 Метрические параметры текста

Структура TEXTMETRICS обеспечивает полную информацию о выбранном в контексте устройства шрифте. Значение tmInternalLeading – это величина пустого

пространства, отведенного для указания специальных знаков над символом. Если это значение равно 0, то помеченные прописные буквы делаются немного меньше, чтобы специальный символ поместился внутри верхней части символа. Значение tmExternalLeading – это величина пустого пространства, которое разработчик шрифта установил для использования между строками символов. В структуре TEXTMETRICS имеется два поля, описывающих ширину символа:

tmAveCharWidth – усредненная ширина символов строки,

tmMaxCharWidth – ширина самого широкого символа шрифта.

Для фиксированного шрифта эти величины одинаковы.

2.9 Форматирование текста

Поскольку размеры системного шрифта не меняются в рамках одного сеанса работы с Windows, необходимо вызывать GetTextMetrics только один раз. Хорошо делать это при обработке сообщения WM_CREATE. Это сообщение первое, которое принимает оконная процедура. Windows вызывает оконную процедуру с сообщением WM_CREATE, когда вызывается в WinMain функция CreateWindow.

Внутри оконной процедуры можно определить две переменные для хранения средней длины символов (cxChar) и полной высоты символов (cyChar):

static int cxChar, cyChar;

Рассмотрим пример обработки сообщения WM_CREATE:

```
case WM_CREATE:
hdc = GetDC (hwnd) ;
GetTextMetrics (hdc, &tm) ;
cxChar = tm.tmAveCharWidth ;
cyChar = tm.tmHeigth + tm.tmExternalLeading ;
ReleaseDC (hwnd, hdc) ;
return 0 ;
```

Если нет нужды учитывать межстрочное пространство, то можно использовать: cyChar = tm.tmHeigth ;

Обычно оставляют пустое поле cyChar в верхней части рабочей области, и cxChar в ее левой части. Для вывода на экран нескольких строк текста с выравниваем па левому краю при вызове функции TextOut используют следующие значение координаты x: cxChar значение координаты y в TextOut: cyChar*(1+i), где i – номер строки начиная с 0.

Программа, которая выводит на экран результаты полученные при вызове функции GetSystemMetrics, приведена в качестве примера.

Пример 2.1 – Файл Sismets.h

```
#define NUMLINES ((int) (sizeof sysmetrics / sizeof sysmetrics [0]))
struct {
   int  iIndex ;
   char *szLabel ;
   char *szDesc ;
}
sysmetrics [] ={
SM_CXSCREEN,      "SM_CXSCREEN",     "Screen width in pixels",
SM_CYSCREEN,      "SM_CYSCREEN",     "Screen height in pixels",
SM_CXVSCROLL,     "SM_CXVSCROLL",    "Vertical scroll arrow width",
SM_CYHSCROLL,     "SM_CYHSCROLL",    "Horizontal scroll arrow height",
SM_CYCAPTION,     "SM_CYCAPTION",    "Caption bar height",
SM_CXBORDER,      "SM_CXBORDER",     "Window border width",
SM_CYBORDER,      "SM_CYBORDER",     "Window border height",
SM_CXDLGFRAME,    "SM_CXDLGFRAME",   "Dialog window frame width",
SM_CYDLGFRAME,    "SM_CYDLGFRAME",   "Dialog window frame height",
SM_CYVTHUMB,      "SM_CYVTHUMB",     "Vertical scroll thumb height",
SM_CXHTHUMB,      "SM_CXHTHUMB",     "Horizontal scroll thumb width",
SM_CXICON,        "SM_CXICON",       "Icon width",
SM_CYICON,        "SM_CYICON",       "Icon height",
SM_CXCURSOR,      "SM_CXCURSOR",     "Cursor width",
SM_CYCURSOR,      "SM_CYCURSOR",     "Cursor height",
SM_CYMENU,        "SM_CYMENU",       "Menu bar height",
SM_CXFULLSCREEN,  "SM_CXFULLSCREEN", "Full screen client area width",
SM_CYFULLSCREEN,  "SM_CYFULLSCREEN", "Full screen client area height",
SM_CYKANJIWINDOW, "SM_CYKANJIWINDOW", "Kanji window height",
SM_MOUSEPRESENT,  "SM_MOUSEPRESENT", "Mouse present flag",
SM_CYVSCROLL,     "SM_CYVSCROLL",    "Vertical scroll arrow height",
SM_CXHSCROLL,     "SM_CXHSCROLL",    "Horizontal scroll arrow width",
SM_DEBUG,         "SM_DEBUG",        "Debug version flag",
SM_SWAPBUTTON,    "SM_SWAPBUTTON",   "Mouse buttons swapped flag",
SM_RESERVED1,     "SM_RESERVED1",    "Reserved",
SM_RESERVED2,     "SM_RESERVED2",    "Reserved",
SM_RESERVED3,     "SM_RESERVED3",    "Reserved",
SM_RESERVED4,     "SM_RESERVED4",    "Reserved",
SM_CXMIN,         "SM_CXMIN",        "Minimum window width",
SM_CYMIN,         "SM_CYMIN",        "Minimum window height",
SM_CXSIZE,        "SM_CXSIZE",       "Minimize/Maximize icon width",
SM_CYSIZE,        "SM_CYSIZE",       "Minimize/Maximize icon height",
SM_CXFRAME,       "SM_CXFRAME",      "Window frame width",
SM_CYFRAME,       "SM_CYFRAME",      "Window frame height",
SM_CXMINTRACK,    "SM_CXMINTRACK",   "Minimum window tracking width",
SM_CYMINTRACK,    "SM_CYMINTRACK",   "Minimum window tracking height",
SM_CXDOUBLECLK,   "SM_CXDOUBLECLK",  "Double click x tolerance",
SM_CYDOUBLECLK,   "SM_CYDOUBLECLK",  "Double click y tolerance",
SM_CXICONSPACING, "SM_CXICONSPACING","Horizontal icon spacing",
SM_CYICONSPACING, "SM_CYICONSPACING","Vertical icon spacing",
SM_MENUDROPALIGNMENT,"SM_MENUDROPALIGNMENT","Left or right menu drop",
SM_PENWINDOWS,    "SM_PENWINDOWS",   "Pen extensions installed",
```

```
SM_DBCSENABLED,   "SM_DBCSENABLED",  "Double-Byte Char Set enabled",
SM_CMOUSEBUTTONS, "SM_CMOUSEBUTTONS","Number of mouse buttons",
SM_SHOWSOUNDS,    "SM_SHOWSOUNDS",   "Present sounds visually"
} ;
```

Пример 2.2 – Файл Sismets1.c

```
#include <windows.h>
#include <string.h>
#include "sysmets.h"
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
     {
     static char szAppName[] = "SysMets1" ;
     HWND        hwnd ;
     MSG         msg ;
     WNDCLASSEX  wndclass ;
     wndclass.cbSize        = sizeof (wndclass) ;
     wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
     wndclass.lpfnWndProc   = WndProc ;
     wndclass.cbClsExtra    = 0 ;
     wndclass.cbWndExtra    = 0 ;
     wndclass.hInstance     = hInstance ;
     wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
     wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
     wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
     wndclass.lpszMenuName  = NULL ;
     wndclass.lpszClassName = szAppName ;
     wndclass.hIconSm       = LoadIcon (NULL, IDI_APPLICATION) ;
     RegisterClassEx (&wndclass) ;
     hwnd = CreateWindow (szAppName, "Get System Metrics No. 1",
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;
     ShowWindow (hwnd, iCmdShow) ;
     UpdateWindow (hwnd) ;
     while (GetMessage (&msg, NULL, 0, 0))
          {  TranslateMessage (&msg) ;
          DispatchMessage (&msg) ;   }
     return msg.wParam ;
     }
LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM
lParam)
     {
     static int  cxChar, cxCaps, cyChar ;
     char        szBuffer[10] ;
     HDC         hdc ;
     int         i ;
     PAINTSTRUCT ps ;
     TEXTMETRIC  tm ;
     switch (iMsg)
          {
```

```
        case WM_CREATE :
             hdc = GetDC (hwnd) ;
             GetTextMetrics (hdc, &tm) ;
             cxChar = tm.tmAveCharWidth ;
             cxCaps = (tm.tmPitchAndFamily & 1 ? 3:2) * cxChar / 2;
             cyChar = tm.tmHeight + tm.tmExternalLeading ;
             ReleaseDC (hwnd, hdc) ;
             return 0 ;
        case WM_PAINT :
             hdc = BeginPaint (hwnd, &ps) ;
             for (i = 0 ; i < NUMLINES ; i++) {
                 TextOut (hdc, cxChar, cyChar * (1 + i),
                    sysmetrics[i].szLabel,
                    strlen (sysmetrics[i].szLabel)) ;
                 TextOut (hdc, cxChar + 22 * cxCaps,
                    cyChar * (1 + i), sysmetrics[i].szDesc,
                    strlen (sysmetrics[i].szDesc)) ;
                 SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;
                 TextOut (hdc, cxChar + 22 * cxCaps + 40 * cxChar,
                    cyChar * (1 + i), szBuffer,
                    wsprintf (szBuffer, "%5d",
                    GetSystemMetrics (sysmetrics[i].iIndex)));
                 SetTextAlign (hdc, TA_LEFT | TA_TOP) ;
             }
             EndPaint (hwnd, &ps) ;
             return 0 ;
        case WM_DESTROY :
             PostQuitMessage (0) ;
             return 0 ;
        }
return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}
```

2.10 Оконная процедура программы

Оконная процедура WndProc обрабатывает в программе 3 сообщения: WM_CREATE, WM_PAINT, WM_DESTROY. При обработке сообщения WM_CREATE SYSMETS1 получает контекст устройства путем вызова GetDC, а также размеры текста вызывая функцию GetTextMetrics. SYSMETS1 также сохраняет среднюю ширину символов верхнего регистра в статической переменной cxCaps. Для фиксированного шрифта cxCaps = cxChar. Для пропорционального шрифта cxCaps = 150% от cxChar. Младший бит поля tmPitchAndFamily структуры TEXTMETRICS равен 1 для пропорционального шрифта и 0 для фиксированного. SYSMETS1 использует значение этого бита следующим образом: cxCaps =(tm.tmPitchAndFamily & 1 ? 3 : 2)*cxChar/2 ; SYSMETS1 полностью выполняет процедуру рисования окна во время обработки сообщения WM_PAINT. В цикле for обрабатываются все элементы

структуры sysmetrics, определенной в SYSMETS.H. Три колонки текста выводятся на экран тремя функциями TextOut. В каждом случае третий параметр TextOut – это выражение: cxChar * (1 + i). Этот параметр указывает в пикселях положение верхней границы строки символов относительно верхней границы рабочей области. Первая инструкция TextOut выводит на экран первую колонку идентификаторов, написанных прописными буквами. Второй параметр TextOut – это cxChar. Он делает отступ от левого края равным одному символу. Текст берется из поля szLabel структуры sysmetrics. Функция strlen используется для получения длины строки. Вторая инструкция TextOut выводит на экран описание значений системных размеров. Эти описания хранятся в поле szDesc структуры sismetrics. Третий параметр в этом случае у TextOut имеет следующий вид:

cxChar + 22 * cxCaps.

Максимальная длина первой колонки равна 20 символам, поэтому вторая колонка должна начинаться как минимум на 20 *cxCaps правее начала первой колонки. Треть инструкция TextOut выводит на экран численные значения, полученные от функции GetTextMetrics. Пропорциональный шрифт делает форматирование колонки, выровненных по правому краю чисел, слегка обманчивым. Каждая цифра от 0 до 9 имеет одну и ту же ширину, но эта ширина больше, чем ширина пробела. Числа могут быть шириной в одну и более цифр, поэтому начальное горизонтальное положение чисел может меняться от строки к строке. Функция SetTextAlign позволяет выводить колонку выровненных по правому краю чисел, задавая положение последней цифры, вместо первой.

SetTextAlign (hdc, TA_RIGHT | TA_TOP);

После этого координаты, переданные TextOut, будут задавать правый верхний угол строки текста вместо ее левого верхнего угла.

2.11 Размер рабочей области

Большинство окон можно развернуть, и рабочая область займет весь экран, за исключением панели заголовка программы. Полный размер этой рабочей области

становится доступным, благодаря функции GetSystemMetrics, использующей параметры SM_CXFULLSCREEN и SM_CYFULLSCREEN. Минимальный размер окна может быть совершенно незначительным, иногда почти невидимым, когда рабочая область практически исчезает. Одним из наиболее общих способов определения размера рабочей области окна является обработка сообщения WM_SIZE в оконной процедуре. Windows посылает в оконную процедуру сообщение WM_SIZE при любом изменении размеров окна. Переменная lParam, преданная в оконную процедуру, содержит ширину рабочей области в млад- шем слове и высоту в старшем слове. Код программы для обработки этого сообщения часто выглядит следующим образом:

```
static int cxClient, cyClient;
[другие инструкции программы]
case WM_SIZE:
  cxClient = LOWORD (lParam);
  cyClient = HIWORD (lParam);
return 0;
```

Макросы LOWORD и HIWORD определяются в заголовочных файлах Windows. Такие как cxChar и cyChar, переменные cxClient и cyClient определяются внутри оконной процедуры в качестве статических, так как они используются позднее при обработке других сообщений. За сообщением WM_SIZE будет следовать WM_PAINT. Потому что при определении класса окна задан стиль класса: CS_HREDRAW | CS_VREDRAW; такой стиль класса указывает на необходимость перерисовки как при горизонтальном так и при вертикальном изменении размеров окна.

2.12 Полосы прокрутки

Полосы прокрутки предназначены для просмотра информации как в вертикальном, так и в горизонтальном направлениях. Вставить в окно приложения вертикальную или горизонтальную полосу прокрутки очень просто. Все, что нужно сделать, это включить идентификатор WS_VSCROLL (вертикальная прокрутка) и WS_HSCROLLW (горизонтальная прокрутка) или оба сразу в описание стиля окна в инструкции CreateWindow. Эти полосы прокрутки всегда размещаются у правого

края или в нижней части окна и занимают всю высоту или ширину рабочей области. Рабочая область не включает в себя пространство, занятое полосами прокрутки. Ширина вертикальной полосы прокрутки и высота горизонтальной постоянны для конкретного дисплейного драйвера. Если необходимы эти значения, их можно получить, вызвав функцию GetSystemMetrics.

2.13 Диапазон и положение полос прокрутки

Каждая полоса прокрутки имеет диапазон и положение бегунка внутри диапазона. Когда бегунок находится в крайней верхней (или крайней левой) части полосы прокрутки, положение бегунка соответствует минимальному значению диапазона. Крайнее правое или крайнее нижнее положение бегунка соответствует максимальному значению диапазона. По умолчанию устанавливается диапазон полос прокрутки от 0 до 100. Но его легко изменить: SetScrollRange(hwnd, iBar, iMin, iMax, bRedraw); параметр iBar равен либо SB_VERT, либо SB_HORZ, iMin и iMax соответствуют диапазону полосы прокрутки. bRedraw устанавливается в TRUE, если не- обходимо, чтобы Windows перерисовывала полосы прокрутки на основе вновь заданного диапазона. Для установки нового положения бегунка используется функция SetScrollPos: SetScrollPos(hwnd, iBar, iPos, bRedraw); iPos – это новое положение бегунка. Для получения текущего диапазона и положения бегунка применяют функции GetScrollRange и GetScrollPos.

2.14 Сообщения полос прокрутки

Windows посылает оконной процедуре асинхронные сообщения WM_VSCROLL и WM_HSCROLL, когда на полосе прокрутки щелкают мышью или перетаскивают бегунок. Каждое действие мыши на полосе прокрутки вызывает как минимум два сообщения, одно при нажатии кнопки, а другое при отпускании ее. Младшее слово параметра wParam, которое объединяет сообщения WM_VSCROLL и WM_HSCROLL – это число, показывающее, что мышь осуществляет какие-то действия на полосе прокрутки. Его значения соответствуют определенным идентификаторам:

Таблица 2.1 – Таблица стилей класса

Идентификатор	Описание
SB_LINEUP	Нажата клавиша мыши на верхней (левой) кнопке со стрелкой в вертикальной (горизонтальной) полосе прокрутки.
SB_ENDSCROLL	Отпущена клавиша мыши на полосе прокрутки.
SB_LINEDOWN	Нажата клавиша мыши на нижней (правой) кнопке со стрелкой в вертикальной (горизонтальной) полосе прокрутки.
SB_PAGEUP	Нажата клавиша мыши между бегунком и верхней (левой) кнопкой со стрелкой в вертикальной (горизонтальной) полосе прокрутки.
SB_PAGEDOWN	Нажата клавиша мыши между бегунком и нижней (правой) кнопкой со стрелкой в вертикальной (горизонтальной) полосе прокрутки.
SB_THUMBTRACK	Нажата клавиша на бегунке.
SB_THUMBPOSITION	Отпущена клавиша на бегунке.

Если младшее слово параметра wParam равно SB_THUMBTRACK или SB_THUMBPOSITION, то старшее слово wParam определяет текущее положение полосы прокрутки. Во всех других случаях wParam игнорируется. Также игнорируется параметр lParam, который используется в окнах диалога. Также существуют значения wParam равные SB_TOP и SB_BOTTOM. Показывающие, что полосы прокрутки переведены в свое максимальное или минимальное положение. Обработка сообщений SB_THUMBTRACK и SB_THUMBPOSITION весьма проблематично. Если установлен большой диапазон полосы прокрутки, а пользователь быстро перемещает бегунок по полосе, то Windows отправит вашей оконной процедуре множество сообщений SB_THUMBTRACK. Программа столкивается с проблемой обработки этих сообщений. Поэтому лучше обрабатывать сообщение SB_THUMBPOSITION, которое означает, что бегунок оставлен в покое.

Однако, если есть возможность быстро обновлять содержимое экрана, обрабатывайте сообщение SB_THUMBTRACK.

2.15 Прокрутка в программе SYSMETS

Обновленный вызов функции CreateWindow добавляет вертикальную полосу прокрутки к окну, благодаря включению в описание стиля ласса идентификатора

WS_VSCROLL: WS_OVERLAPPEDWINDOW | WS_VSCROLL. Приводится только оконная процедура программы.

```
LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM
lParam)
     {
     static int  cxChar, cxCaps, cyChar, cyClient, iVscrollPos ;
     char        szBuffer[10] ;
     HDC         hdc ;
     int         i, y ;
     PAINTSTRUCT ps ;
     TEXTMETRIC  tm ;
     switch (iMsg)    {
          case WM_CREATE :
               hdc = GetDC (hwnd) ;
               GetTextMetrics (hdc, &tm) ;
               cxChar = tm.tmAveCharWidth ;
               cxCaps = (tm.tmPitchAndFamily & 1 ? 3 : 2)*cxChar / 2 ;
               cyChar = tm.tmHeight + tm.tmExternalLeading ;
               ReleaseDC (hwnd, hdc) ;
               SetScrollRange (hwnd, SB_VERT, 0, NUMLINES, FALSE) ;
               SetScrollPos   (hwnd, SB_VERT, iVscrollPos, TRUE) ;
               return 0 ;
          case WM_SIZE :
               cyClient = HIWORD (lParam) ;
               return 0 ;
          case WM_VSCROLL :
               switch (LOWORD (wParam))
                    {
                    case SB_LINEUP :
                         iVscrollPos -= 1 ;
                         break ;
                    case SB_LINEDOWN :
                         iVscrollPos += 1 ;
                         break ;
                    case SB_PAGEUP :
                         iVscrollPos -= cyClient / cyChar ;
                         break ;
                    case SB_PAGEDOWN :
```

```
                    iVscrollPos += cyClient / cyChar ;
                    break ;
               case SB_THUMBPOSITION :
                    iVscrollPos = HIWORD (wParam) ;
                    break ;
               default :
                    break ;
               }
          iVscrollPos = max (0, min (iVscrollPos, NUMLINES)) ;
          if (iVscrollPos != GetScrollPos (hwnd, SB_VERT))
               {
               SetScrollPos (hwnd, SB_VERT, iVscrollPos, TRUE) ;
               InvalidateRect (hwnd, NULL, TRUE) ;
               }
          return 0 ;
     case WM_PAINT :
          hdc = BeginPaint (hwnd, &ps) ;

          for (i = 0 ; i < NUMLINES ; i++)
               {
               y = cyChar * (1 - iVscrollPos + i) ;
               TextOut (hdc, cxChar, y, sysmetrics[i].szLabel,
strlen (sysmetrics[i].szLabel));
               TextOut (hdc, cxChar + 22 * cxCaps, y,
sysmetrics[i].szDesc,
 strlen (sysmetrics[i].szDesc)) ;
               SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;
               TextOut (hdc, cxChar + 22 * cxCaps + 40 * cxChar,
y, szBuffer,
                         wsprintf (szBuffer,
"%5d",GetSystemMetrics (sysmetrics[i].iIndex))) ;
               SetTextAlign (hdc, TA_LEFT | TA_TOP) ;
               }
          EndPaint (hwnd, &ps) ;
          return 0 ;
     case WM_DESTROY :
          PostQuitMessage (0) ;
          return 0 ;
     }
return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}
```

К оконной процедуре WndProc добавляются две строки для установки диапазона и положения полосы прокрутки во время обработки сообщения WM_CREATE:

SetScrollRange (hwnd, SB_VERT, 0, NUMLINES, FALSE);

SetScrollPass (hwnd, SB_VERT, iVscrollPos, TRUE);

Т.к. структура sysmetrics содержит NUMLINES строк, то диапазон полосы прокрутки устанавливается от 0 до NUMLINES. Каждое положение бегунка соответствует строке текста в верхней части рабочей области.

Если положение бегунка равно 0, то в окне сверху будет пустая строка, если – NUMLINES, то последняя строка.

2.16 Функция ScrollWindow

Для каждого действия с полосой прокрутки необходимо сначала рассчитать приращение ее текущей позиции при обработке сообщения WM_VSCROLL и WM_HSCROLL. Это значение затем можно использовать для прокрутки имеющегося в окне содержимого с помощью вызова функции ScrollWindow. Функция ScrollWindow листает содержание клиентской области определенного окна. Эта функция существует для совместимости с Win16. Для новых приложений, используют функцию ScrollWindowEx.

BOOL ScrollWindow(HWND hWnd, int XAmount, int YAmount,

CONST RECT *lpRect, CONST RECT *lpClipRect);

hWnd – Идентифицирует окно, где клиентская область должна быть пролистана;

XAmount, YAmount – величины прокрутки в пикселя;

lpRect –Указатель на структуру RECT, определяющую блок клиентской области, которая будет пролистана. Если этот параметр NULL, вся клиентская область будет пролистана. lpClipRect – Указатель на структуру RECT, содержащую координаты прямоугольника отсечения. Воздействуют только на биты устройства внутри прямоугольника отсечения. Если функция выполнилась без ошибки, возвращаемое значение отлично от нуля. При сбое, возвращаемое значение - NULL. Windows делает недействительным прямоугольную зону рабочей области, открываемую операцией прокрутки. Это приводит к выдаче сообщения WM_PAIT.

InvalidateRect больше не нужна. Функция ScrollWindow не является процедурой GDI, поэтому ей не нужен описатель контекста устройства.

int ScrollWindowEx(HWND hWnd, int dx, int dy, CONST RECT *prcScroll, CONST RECT *prcClip, HRGN hrgnUpdate, LPRECT prcUpdate, UINT flags);

dx, dy – величины прокрутки в пикселях;

PrcScroll – указатель на структуру RECT, определяющую блок клиентской области, которая будет пролистана. Если этот параметр NULL, вся клиентская область пролистана.

PrcClip –указатель на структуру RECT, содержащую координаты прямоугольника отсечения.

HrgnUpdate – Идентифицирует область, которая изменяется, для объявления области прокрутки недействительной. Этот параметр может быть NULL.

PrcUpdate – указатель на структуру RECT, получающую границы недействительного прямоугольника. Этот параметр может быть NULL.

flags – Определяет флажки, которые управляют прокруткой. Этот параметр может быть одно из следующих значений:

SW_ERASE – Стирает недействительную область, посылая сообщение WM_ERASEBKGND к окну, определяется с флажком

SW_INVALIDATE;

SW_INVALIDATE – Объявляет область недействительной, идентифицированную hrgnUpdate параметром после прокрутки.

SW_SCROLLCHILDREN – Листает все дочерние окна, которые пересекают прямоугольник, указанный на prcScroll параметром. Дочерние окна пролистаны числом пикселов, определенных dx и dy параметрами.

Windows посылает WM_MOVE сообщение всем дочерним окнам, которые пересекают prcScroll прямоугольник.

3. Контрольные вопросы

1) При каких ситуациях возникает сообщение WM_PAINT?

2) Что такое действительные и недействительные прямоугольники?

3) Что такое контекст устройства и для чего он нужен?

4) Опишите способы получения контекста устройства.

5) Опишите функцию TextOut.

6) Что такое системный шрифт и где он применяется?

7) Для чего нужны полосы прокрутки?

8) Как вставить в рабочую область окна вертикальную и горизонтальную полосы прокрутки?

9) Какие сообщения вырабатывают полосы прокрутки?

10) Как задаётся диапазон полосы прокрутки и чему он равен изначально?

4. Задание

1) Изучить описание лабораторной работы.

2) Написать программу, которая получает контекст устройства.

3) Добавьте в рабочую область окна написанной программы текст, который превышает размеры окна по вертикали и горизонтали.

4) Отладить и протестировать полученную программу.

5) Добавьте в программу вертикальную и горизонтальную полосы прокрутки.

6) Отладить и протестировать полученную программу.

7) Написать программу в которой при передвижении бегунка текст содержащийся в окне двигался вместе с ним.

8) Отладить и протестировать полученную программу.

9) Оформить отчёт.

Список литературы

1. Гордеев А. В., Молчанов А. Ю. Системное программное обеспечение. –СПб.: Питер, 2001. – 736 с.: ил.

2. Харт, Д. М. Системное программирование в среде Win32, 2-е изд.: Пер.с англ.: – М.: Издательский дом «Вильямс», 2001.– 464 с.: ил. – Парал.тит. англ.

3. Петзольд Ч. Программирование для Windows 95; в двух томах. Том 1/Пер. С англ. — СПб.: BHV— Санкт-Петербург, 1997.— 752 с.: ил.
4. Петзольд Ч. Программирование для Windows 95; в двух томах. Том 2/Пер. С англ. — СПб.: BHV— Санкт-Петербург, 1997.— 368 с.: ил.
5. Румянцев П.В. Азбука программирования в Win 32 API. – М.: Горячая Линия - Телеком, 2004. – 312 с.
6. Ганеев Р. М. Проектирование интерфейса пользователя средствами Win32 API. – М: Горячая Линия – Телеком, 2001. – 336 с.
7. Харт, Д. М. Системное программирование в среде Windows Windows System Programming. – М.: Вильямс, 2005.– 336 с.: ил.
8. Неббет, Гэри. Справочник по базовым функциям API Windows NT/2000 Windows NT/2000 – М: Вильямс, 2002. – 528 с.