

Windows System Programming

CS462 – SA Notes
Dr. Mohammed Ameen

Introduction

In old days, the operating system program executes the program's instructions in sequence one instruction after another as they arranged by the programmer. When the program calls a function or interacts with the Operating System (OS) by calling one of the OS's service functions, the function's instructions executed in similar way and the execution continue sequentially until the end of the last instruction of the main program. In this traditional way of programming the program only need one main function to start the execution. Nowadays, in event driven programming environment such as Windows, the interaction between the program and the OS is a message-based interaction. In this interaction the program start in a wait mode until the user create an event like for example, press a key or click a mouse button. The OS generates appropriate message and inserts it in application's message queue. The program, when it ready, should dispatch the messages (read a message from the queue and send it back to the OS) one message after another for time scheduling purpose. After that, the OS then send the message again to the program and the program respond by taking appropriate actions. In this kind of interaction, a simple program need at least two functions, the main function that start the message loop (WinMain()) and the other called by the OS (Windows Function()) and respond to the passed message. The following is a skeleton program for Windows operating system that can be used as a starter for studying and testing windows applications.

Windows Skeleton Program

```
WinMain ( )  
{  
    Define a window structure  
    Register the windowstructure  
    Create the window  
    Display the window  
    Run the message loop  
}
```

```
WindowFunc ( )  
{  
    Switch ( message )  
    {  
        case message 1 :  
        ----  
        ----  
        ----  
        case message 2 :  
        ----  
        ----  
        ----  
        default :  
        ----  
        ----  
        ----  
    }  
}
```

```

/* A minimal windows skeleton program. */

#include <windows.h>

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

/* name of windows class */
char szWinName[] = "MyWin";

int WINAPI WinMain(HINSTANCE hThisInst, HINSTANCE hPrevInst, LPSTR
                  lpszArgs, int nWinMode)
{
    HWND          hwnd;
    MSG           msg;
    WNDCLASSEX    wcl;

    /* Define a window class. */
    wcl.cbSize = sizeof(WNDCLASSEX);

    wcl.hInstance = hThisInst; /* handle to this instance */

    wcl.lpszClassName = szWinName; /* window class name */

    wcl.lpfnWndProc = WindowFunc; /* window function */

    wcl.style = 0; /* default style */

    wcl.hIcon = LoadIcon(NULL, IDI_APPLICATION); /*std icon */
    wcl.hIconSm = LoadIcon(NULL, IDI_WINLOGO); /*small icon */
    wcl.hCursor = LoadCursor(NULL, IDC_ARROW); /*cursor style */

    wcl.lpszMenuName = NULL; /* no menu */

    wcl.cbClsExtra = 0; /* no extra */

    wcl.cbWndExtra = 0; /* information needed */

    /* Make the window background white. */
    wcl.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);

    /* Register the window class */
    if(!RegisterClassEx(&wcl)) return 0;

```

```

/* Now that a window class has been registered,
** a window can be created. */
hwnd = CreateWindow(
    szWinName,          /* name of window class */
    "Windows Skeleton Program", /* title */
    WS_OVERLAPPEDWINDOW, /* window style:normal */
    CW_USEDEFAULT,     /* X coordinate - let windows decide */
    CW_USEDEFAULT,     /* Y coordinate - let windows decide */
    CW_USEDEFAULT,     /* width - let windows decide */
    CW_USEDEFAULT,     /* height- let windows decide */
    HWND_DESKTOP,      /* no parent window */
    NULL,
    hThisInst, /*handle of this instance of hte program */
    NULL /* no additional arguments */
);

/* Display the window. */
ShowWindow(hwnd, nWinMode);
UpdateWindow(hwnd);

/* Create the message loop. */
while(GetMessage(&msg, NULL, 0, 0))
{
    TranslateMessage(&msg); /* allow use of keyboard */
    DispatchMessage(&msg); /*return control to windows */
}
return msg.wParam;
}

/* This function is called by Windows OS and
** is passed message from the message queue.
*****/

LRESULT CALLBACK WindowFunc(HWND hwnd, UINT message, WPARAM wParam,
                             LPARAM lParam)
{
    switch(message) {
        case WM_DESTROY: /* terminate the program */
            PostQuitMessage(0);
            break;
        default:
            /* Let Windows NT process any messages
            ** not specified in the preceding switch
            ** statement. */
            return DefWindowProc(hwnd, message, wParam, lParam);
    }
    return 0;
}

```

The WinMain function starts by creating a window since we need one to display the output. Unlike the old programming under DOS, there is no standard output. After a window created the function start the message loop. The second function usually called the Windows Function or Windows Procedure has a switch statement to handle different messages.

WINDOWS.H header file

- must be included by all windows programs

```
#include <windows.h>
```

- Contains the API function prototypes and data type definitions.
- some windows data types
 - Handle data type
 - HANDLE is a 32-bit integer identifies some resources.
 - there are many handle types, but they are all same
 - HWND is used as a window handle.
 - HINSTANCE is used as a program handle.
 - All handles begin with an H.
 - Basic data types
 - BYTE is an 8-bit unsigned character
 - WORD is a 16-bit unsigned integer
 - DWORD is an unsigned long integer
 - LONG is another name for DWORD
 - UINT is an unsigned 32-bit integer (16-bit if compiled under windows 3.1)
 - BOOL is an integer to represent true or false
 - LPSTR is a pointer to a string
 - LPCSTR is a constant pointer to a string
 - Structured data type
 - MSG holds a window message
 - WNDCLASSEX is a structure that defines a window style.

WinMain function

- is where a windows program begin execution
- must be compiled using WINAPI or APIENTRY calling convention (the default is C\C++ calling convention. another an old calling convention that is used in Windows 3.1 is PASCAL)
- Return type should be int.
- is passed four parameters
 - *hThisInst*: refers to the current instance of the program. Since windows operating system is a multitasking, it is possible that more than one instance of your program may be running at same time.
 - *hPrevInst*: NULL or zero if no instance of a program is running.
 - *lpszArgs*: pointer to a string that holds any command line arguments (include the program name)
 - *nWinMode*: contains an integer value define how the window will be displayed when the program begins execution. four possible values defined in windows.h header file:
 - SW_HIDE: remove the window
 - SW_MINIMIZE
 - SW_MAXIMIZE
 - SW_RESTORE: return a window to normal size.

Defining a window structure

- in order to create a window, a data structure of type `WNDCLASSEX` must be created by the following statement:

```
WNDCLASSEX    wcl;
```

- Next, the fields of the defined window structure, `wcl`, must be filled.
- The field ***cbSize*** is assigned the size of `WNDCLASSEX` structure size using the ***sizeof*** operator.

```
wcl.cbSize = sizeof (WNDCLASSEX);
```

- The field ***hInstance*** is assigned the program instance handle that passed as an argument to `WinMain` function.

```
wcl.hInstance = hThisInst;
```

- The field ***lpszClassName*** point to the name of window class structure which are stored in ***szWinName*** variable.

```
wcl.lpszClassName = szWinName;
```

- The field ***lpfnWndProc*** point to the windows function.

```
wcl.lpfnWndProc = WindowFunc;
```

- The field ***style*** is assigned zero and that means: no style is specified.

```
wcl.style = 0; /* default style */
```

- Defining resources

- All windows program use API to define resources such as mouse, cursor, icons, and menu.
- A program may define its own version of these resources or it may use the built-in resources as the skeleton does.
- handle to these resources must be assigned to appropriate fields of `WNDCLASSEX` structure

- load the icon
 - A windows application has two icons, standard icon and small icon.
 - Small icon (16x16 bitmap) is used when the application is minimized and as system menu.
 - The standard icon (32x32) is used when you move the application to desktop.
 - You can load the icon in the WINDCLASSEX fields, ***hIcon*** and ***hIconSm*** by calling the API function ***LoadIcon***.

```
wcl.hIcon = LoadIcon(NULL, IDI_APPLICATION);
wcl.hIconSm= LoadIcon(NULL, IDI_WINLOGO);
```

- the prototype for ***LoadIcon ()*** is

```
HICON LoadIcon( HINSTANCE hInst, LPCSTR lpszName);
```

- These function returns a handle to an icon, ***hInst*** specify the handle of the module that contain the icon. ***lpszName*** specifies the icon's name.
- to use the built-in icons, use NULL for ***hInst*** and choose one of the following macros for icon name

Icon Macro	Shape
IDI_APPLICATION	Default icon
IDI_ASTERISK	Information icon
IDI_EXCLAMATION	Exclamation point icon
IDI_HAND	Stop sign
IDI_QUESTION	Question mark icon
IDI_WINLOGO	Windows Logo

- Load mouse cursor
 - To load the mouse cursor use the API function ***LoadCursor()*** which is similar to ***LoadIcon()*** idea.

```
wcl.hCursor= LoadCursor(NULL, IDC_ARROW);
```

- to use the built-in cursor, use NULL for *hInst* and choose one of the following macros for cursor name

Cursor Macro	Shape
IDC_ARROW	Default arrow pointer
IDC_CROSS	Cross hairs
IDC_IBEAM	Vertical I-beam
IDC_WAIT	Hourglass

- The field *lpszMenuName* is assigned NULL which means no menu is specified.

```
wcl.lpszMenuName = NULL; /* no menu */
```

- No extra information is needed by assign zero to the fields, *cbClsExtra* and *cbWndExtra*.

```
wcl.cbClsExtra = 0;
wcl.cbWndExtra = 0;
```

- Load the brush
 - A brush is a resource that paints the screen.
 - To obtain a brush handle, use the API *GetStockObject()*.

```
wcl.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
```

- *GetStockObject()* can also be used to obtain handle to one of the others standard display objects such as pens and fonts.
- the prototype is as the following

```
HGDIOBJ GetStockObject (int object);
```

- The function is passed an integer value and returns a handle to the object specified by object. The type *HGDIOBJ* is a *GDI* handle.
- The following are some built-in brushes macros that can be used to obtain a brush.

Macro name	Background type
BLACK_BRUSH	BLACK
DKGRAY_BRUSH	Dark gray
HOLLOW_BRUSH	Light gray
WHITE_BRUSH	White

Registering the window

- Once the data structure of the window has been fully specified, it must be registered.
- The window structure is registered in the operating system by its address.
- The API function *RegisterClassEx*() is passed a pointer to the WNDCLASSEX structure and returns a unique value that identifies the window structure.

```
ATOM RegisterClassEx(CONST WNDCLASSEX *lpWClass);
```

- the *ATOM* is a typedef that is similar to *WORD*. the *lpWClass* is a pointer to the window data structure.

Creating the window

- after defining and registering the window structure, you can use the API function *CreateWindow*() as in the following

```
hwnd = CreateWindow(
    szWinName,          /* name of window class */
    "Windows Skeleton Program", /* title */
    WS_OVERLAPPEDWINDOW, /* window style:normal */
    CW_USEDEFAULT, /* X coordinate - let windows decide */
    CW_USEDEFAULT, /* Y coordinate - let windows decide */
    CW_USEDEFAULT, /* width - let windows decide */
    CW_USEDEFAULT, /* height- let windows decide */
    HWND_DESKTOP, /* no parent window */
    NULL,
    hThisInst, /*handle of this instance of hte program */
    NULL      /* no additional arguments */
);
```

- The function returns the handle of the window it creates or NULL if it failed.

Display the window

- to display the created window, we use the API function *ShowWindow()* whose prototype as in the following:

```
BOOL ShowWindow(HWND hwnd, int nHow)
```

- The function passed the handle of the window to display, *hwnd*, and the display mode is specified in *nHow*.
- The mode in which the window displayed the first time can be specified in the last parameter of *WinMain()*'s *nWinMode*.
- the following are some built-in display macros that can be used

Display Macro	Effect
SW_HIDE	Removes the window
SW_MINIMIZE	Minimizes the window into an icon
SW_MAXIMIZE	Maximizes the window
SW_RESTORE	Returns a window to normal size

- the API function *ShowWindow()* returns zero if the window was not displayed, and returns nonzero if the window was displayed.
- the API function *UpdateWindow()* tells Windows OS to send a message to the application to tell the application program that the main window need to be updated.

The Message Loop

- The last task in *WinMain*() is to start the message loop. When an application is running, it receives messages randomly stored in its message queue by OS until they can be read and processed. A message is read when the application program is ready to process one.
- Your program can read a message from the queue by the API function *GetMessage*(), which has the following prototype:

```
BOOL GetMessage(LPMSG msg, HWND hwnd, UINT min, UINT max);
```

- you will want to pass a pointer to the message data structure *MSG*, shown here

```
typedef struct tagMSG {  
    HWND    hwnd;        /* window that message is for */  
    UINT    message;     /* message */  
    WPARAM  wParam;     /* message-dependent info */  
    LPARAM  lParam;     /* more info */  
    DWORD   time;       /* time message posted */  
    POINT   pt;         /* X, Y location of mouse */  
} MSG;
```

- The *hwnd* field contains a handle to the window which the message is belong to.
- The message itself is stored in *message* field.
- The *wParam* and *lParam* are assigned additional information related to the message, for example the scan code and ASCII code. The *WPARAM* is a typedef for *UINT* and *LPARAM* is a typedef for *LONG*.
- If the message queue is empty then a call for *GetMessage*() will return control to the OS.
- The *GetMessage*() will always return nonzero and that keep the loop up running until the user terminate the program which cause the function return zero and terminate the while loop.
- Inside the loop, the API *TranslateMessage*() will translate the scan code value into ASCII code value for WM_CHAR message that generated when a user press a key.
- The other function inside the loop is the API *DispatchMessage*() function which send the processed message to the OS.

The Window Function

- The Window Function is the second function in the skeleton program that called by windows, not by your program.
- The Window Function is passed messages by the OS and must be not called by your program.
- The **CALLBACK** calling convention is used with this function to tell the compiler that this function will be called by the OS.
- **LRESULT** is a typedef for long integer.

```
LRESULT CALLBACK WindowFunc(HWND hwnd, UINT message,  
                             WPARAM wParam, LPARAM lParam)
```

```
{  
    switch(message) {  
        case WM_DESTROY:           /* terminate the program */  
            PostQuitMessage(0);  
            break;  
        default:  
            /* Let Windows NT process any messages  
            ** not specified in the preceding switch  
            ** statement. */  
            return DefWindowProc(hwnd, message, wParam,  
                                   lParam);  
    }  
    return 0;  
}
```

- this function receives the first four fields in the **MSG** structure.
- The skeleton program responds to only one message, the **WM_DESTROY**.
- When the user terminates the program by pressing the x button at the title bar, the operating system passes **WM_DISTROY** to the Window Function. Your program should respond by calling the API function, **PostQuitMessage()**.
- The **PostQuitMessage** will causes a **WM_QUIT** message to be sent to your program, which causes the **GetMessage()** to return false and stop the message loop.
- The argument of **PostQuitMessage()** is an exit code that is returned in **msg.wParam** inside **WinMain()**.
- All messages not processed by the Window Function (like **WM_QUIT** above) are passed to the OS by **DefWindowProc()** for default processing.