LAB 6, PART 2

A WIN32 GUI APPLICATION

This lab continues the project you started on Monday. You will ensure that your application terminates when the window is closed, and then you will add a File menu and a File > Exit menu item.

Directions. At the top of your .asm file, include a comment with the answers to the questions marked with a ? symbol from **both** Part 1 and Part 2 of this lab. Submit your final .asm file in Canvas.

4. ADDING A WINDOW PROCEDURE (CALLBACK)

In Part 1 of this lab, you added a message loop to your main procedure. Although you could handle messages (button clicks, key presses, etc.) there, this is impractical for large applications.

Instead, when you register a window class, you can create a *window procedure* that handles messages only for that specific type of window. That way, every window's messages can be handled by a different window procedure.

- ❑ ? **The last part of your message loop was a call to DispatchMessage. Look up the documentation for DispatchMessage on MSDN. What does it do?**

Now, you will add a window procedure that handles messages for your application's main window.

- ❑ Start with your project from last time.
- ❑ At the bottom of the .asm file, *before* the END directive, add a procedure called MainWindowProc@16. This will be a procedure that takes four arguments and uses the STDCALL calling convention:

```
MainWindowProc@16 PROC
    hWnd   EQU DWORD PTR [ebp+8]
    uMsg   EQU DWORD PTR [ebp+12]
    wParam EQU DWORD PTR [ebp+16]
    lParam EQU DWORD PTR [ebp+20]

    enter 0, 0

    push lParam
    push wParam
    push uMsg
    push hWnd
    call DefWindowProcA

    leave
    ret 16
MainWindowProc@16 ENDP
```

DefWindowProcA is the default window procedure, which is part of the Windows API. Essentially, MainWindowProc@16 is simply passing on messages to the default window procedure.

❏ ⓘ **Look up the documentation for __stdcall on MSDN. Why did we name this function MainWindowProc@16 instead of, say, MainWindowProc@4 or MainWindowProc@8?**

A window procedure is an example of a *callback function.* Your application never calls it directly. Instead, you pass the address of the function when you register your window class, and Windows invokes your function at an appropriate time (specifically, in response to DispatchMessage).

❏ The .data section included the following variables, which were used when you registered your window class.

```
WNDCLASS_START = $
cbSize        DWORD WNDCLASS_END - WNDCLASS_START
style         DWORD 0
lpfnWndProc   DWORD OFFSET DefWindowProcA
cbClsExtra    DWORD 0
...
```

Change lpfnWndProc so that it stores the address of your new MainWindowProc@16 procedure instead:

```
lpfnWndProc   DWORD OFFSET MainWindowProc@16
```

❏ Run your application again. It should behave exactly the same as before.

5. Handling WM_DESTROY

When you completed Part 1 of this lab, you had created a window, but when you closed it the application continued running—you had to terminate the application manually in the Visual Studio debugger. Now, you will modify the window procedure to terminate your application when the window is closed.

❏ ⓘ **Look up the MSDN documentation for the WM_DESTROY message. When is this message sent?**

❏ There are several symbolic constants defined at the top of your .asm file. Add a symbolic constant for WM_DESTROY:

```
WM_DESTROY = 2h
```

❏ There are also several API function prototypes at the top of your .asm file. Add a prototype for PostQuitMessage

```
PostQuitMessage PROTO,
    nExitCode:DWORD
```

❏ Currently, the MainWindowProc@16 procedure simply calls DefWindowProcA. Change it so it behaves as follows:

- If uMsg == WM_DESTROY, invoke `PostQuitMessage(0)`, and then return 0 in EAX
- Otherwise, invoke `DefWindowProcA(hWnd, uMsg, wParam, lParam)` and return its result in EAX, as before.

❏ Run your program. When you close the window, your program should terminate immediately.

6. Adding a Menu Bar

- ❑ Add constants for MF_POPUP and MF_STRING:

```
MF_POPUP = 10h
MF_STRING = 0h
```

- ❑ Add prototypes for CreateMenu and AppendMenuA:

```
CreateMenu PROTO

AppendMenuA PROTO,
    hMenu:DWORD,
    uFlags:DWORD,
    uIDNewItem:DWORD,
    lpNewItem:DWORD
```

- ❑ In the .data section, add strings for the File and File > Exit menu items:

```
szFile BYTE "&File", 0
szExit BYTE "E&xit", 0
```

- ❑ In the .data section, add DWORDs to store handles for the menu bar and the File menu:

```
hMenu     DWORD ?
hMenuFile DWORD ?
```

- ❑ After the code to register the window class, add code to create a menu structure for your window:

```
; Create the menu bar
call CreateMenu
mov hMenu, eax

; Create the File menu
call CreateMenu
mov hMenuFile, eax

; Add an Exit item to the File menu
push OFFSET szExit
push 0
push MF_STRING
push hMenuFile
call AppendMenuA

; Add the File menu to the menu bar
push OFFSET szFile
push hMenuFile
push MF_POPUP
push hMenu
call AppendMenuA
```

- ❑ Both CreateMenu and AppendMenuA return 0 upon failure. Modify the above code to jump to `error` if any of the CreateMenu or AppendMenuA calls fail.

❑ Change the third-to-last argument to CreateWindowEx so that it uses hMenu for the window's menu bar:

```
; Create a 500x100-pixel window
push 0
push hInstance
push hMenu
push 0
```

❑ Run your application. It should have a File menu with a menu item named Exit. However, when you click on it, nothing should happen.

❑ Run your application again. Hold down the Alt key, and notice that the letter F in "File" is underlined. If you press Alt+F, and hold the Alt key, the x in "Exit" is underlined. So, pressing Alt+F X will exit your application.

❑ ❓ **Look at the definitions of the szFile and szExit variables. How did you specify which letters to underline?**

7. Handling WM_COMMAND

❑ ❓ **Look up the MSDN documentation for the WM_COMMAND message. When is this message sent?**

❑ Add constants for WM_COMMAND:

```
WM_COMMAND = 111h
```

❑ Modify MainWindowProc@16 to behave as follows:

- If uMsg == WM_COMMAND, do the following:
  - Display a message box saying "Goodbye"
  - Invoke `PostQuitMessage(0)`
  - Return 0 in EAX
- If uMsg == WM_DESTROY:
  - Invoke `PostQuitMessage(0)`
  - Return 0 in EAX
- Otherwise, invoke `DefWindowProcA(hWnd, uMsg, wParam, lParam)` and return its result in EAX, as before.

❑ Run your application. Make sure it behaves as follows:

- If you click File > Exit, it should display a message box saying "Goodbye," and then terminate.
- If you close the window by clicking the X in the upper-right corner, the application should terminate without saying "Goodbye."

❑ Make sure the answers to the questions marked with a ❓ are in a comment at the top of your .asm file. Include the answers from Part 1 *and* Part 2 of this lab. Then, submit your completed code in Canvas.