Lecture

Review of Last Lecture

- DLL's
- Processes
- Threads
- Memory Management

The Import Libraries .LIB

- Concept of an import library .LIB
- Import Library is statically linked
- How to create an import library?
- Important System DLLs: Kernel32.dll, User32.dll, Gdi32.dll
- Import Libraries of System DLLs: Kernel32.lib, User32.lib, Gdi32.lib

Variable scope in DLLs

- Static variables have scope limited to the block in which they are declared. As a result, each process has its own instance of the DLL global and static variables by default.

Variable scope in DLLs

Sharing variables across multiple processes

- `#pragma data_seg("OUR_DATA")` and its usage

Specifying a section in the executable in Visual C++ 6.0

- Using linker option:

```
/section: OUR_DATA,rws
```

- Using pre-processor directive in source file:

```
#pragma comment(linker, "/SECTION: OUR_DATA, RWS")
```

Tip: "Quick View" utility can be used to verify shared section(s)

Calling Conventions and DLLs

- Significance of Calling Conventions of the caller and the called function in a DLL
- C++ uses same calling convention / parameter passing as C, but performs name decoration
- extern "C" { … … function declarations … } prevents C++ name-decoration

Windows Hooks

- Win32 Hooks are a way to trap messages before they reach *a single* or *any* application
  - Application level hook
    - Hook that affects a single application, i.e. the one that installs it. The hook procedure can be in the same executable as the original application.
  - System level hook
    - Hook that effects *all* applications in the system. This must be implemented separately in a DLL. This DLL is then loaded in the memory space of every running process, by the system.

Resource-only DLLs

- Including resources in a DLL
- LoadBitmap(), LoadString(), LoadMenu(), LoadIcon()

```
HRSRC FindResource(
  HMODULE hModule, // module handle
  LPCTSTR lpName,  // resource name
  LPCTSTR lpType   // resource type
);
```

`lpType` can be `RT_ICON`, `RT_MENU`, `RT_STRING` etc.

- Use of resource-only DLLs for internationalisation

DLL versions

- Why versioning?
  The comctl32.dll example
- The VERSIONINFO resource statement

VERSIONINFO resource statement

```
VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,1
 PRODUCTVERSION 1,0,0,1
 FILEFLAGSMASK 0x3fL
 FILEFLAGS 0x1L
 FILEOS 0x40004L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904b0"
        BEGIN
                VALUE "CompanyName", "Virtual University\0"
         VALUE "LegalCopyright", "Copyright © 2002 Virtual University\0"
      END
      ... ... ... ...
    END
END
```

DLL versions

```
BOOL GetFileVersionInfo(
  LPTSTR lptstrFilename,   // file name
  DWORD dwHandle,          // ignored
  DWORD dwLen,             // size of buffer
  LPVOID lpData            // version
  information buffer
);
```

- `HRESULT CALLBACK DllGetVersion(`
  `DLLVERSIONINFO *pdvi);`

Thread basics

- WinMain() is the primary thread.
- All other threads created in the process are secondary threads
- Usage of threads

Thread basics

- User Interface (UI) and Worker threads
  - User interface threads own one or more Windows and have their own message queue
  - Worker threads do not own any windows, and may not have a message queue

Thread basics

- Relationship of a Process and threads; a Thread and windows in Windows
  - A process may consist of one or more threads
  - A thread may own zero or more windows

Creating Secondary Threads

- How to create secondary threads?
  - _beginthread() C runtime
  - The CreateThread() Win32 API

Creating Secondary Threads

unsigned long _beginthread(

void(__cdecl * *start_address* **)(void *),**
pointer to thread function

unsigned *stack_size*,
stack-size for new thread

void * *arglist* **);**
argument list to be passed to thread

Threads and message queuing

- The message routing:
  System message queue > Thread message queue > Window procedure (if a UI thread)
- Threads with and without a message queue
- UI threads always have a message queue
- When is the message queue created?
  - The system creates a thread-specific message queue only when the thread makes its first call to one of the Win32 User or GDI functions

The Thread Procedure

Thread procedure can be any function with the following signatures (return value and parameter list):

```
DWORD WINAPI ThreadProc(
   LPVOID lpParameter
);
```

Thread data passed to the thread function

Creating Secondary Threads

```
HANDLE CreateThread(
  LPSECURITY_ATTRIBUTES lpThreadAttributes,// Security Descriptor
  DWORD dwStackSize,                       // initial stack size
  LPTHREAD_START_ROUTINE lpStartAddress,   // thread function
  LPVOID lpParameter,                      // thread argument
  DWORD dwCreationFlags,                   // creation option
  LPDWORD lpThreadId                       // thread identifier
);
```

Secondary Thread Procedure

- The thread functions
- A thread terminates when its thread procedure returns
- Terminating a thread within a thread function: _endthread() and ExitThread()
- Difference between return values of C runtime and Win32 thread creation functions
- The Thread Handles

More about Threads

- Thread handles and thread IDs
- CREATE_SUSPENDED flag
- SuspendThread() , ResumeThread()
- Sleep() call

- Advantages of Threads