Lecture

24

Review of Last Lecture

- Windows Common Controls
- Common controls library commctl32.dll
- InitCommonControlsEx()
- Image Lists
- List View control

Memory management basics

- Separate address space for a process
- Physical and Virtual Memory
- Page table
- Address Space: Linear and Physical Address
- Committed and reserved pages
- Same address in different processes may map to different things or nothing

Basic concepts and today's topics

- What is a Process?
  A running application that consists of a private virtual address space, code, data, and other operating-system resources, such as files, pipes, and synchronization objects that are visible to the process. A process also contains one or more threads that run in the context of the process

What is a Thread ?

- What is a Thread?
  A thread is basically a path of execution through a program. It is also the smallest unit of execution that Win32 schedules. A thread consists of a stack, the state of the CPU registers, and an entry in the execution list maintained by Windows. Each thread shares all of the process's resources.

contd…

What is a Thread ? *contd…*

- A Thread and a Process

  A process consists of one or more threads and the code, data, and other resources of a program in memory. Each thread in a process operates independently. Unless you make them visible to each other, the threads execute individually and are unaware of the other threads in a process.

Linking and the Compiled Code

- What is compiled .OBJ code?
- The purpose of the linker
- Static Linking
- Dynamic Linking
- Dynamic Link Libraries (DLLs)

Dynamic Link Libraries

- Why a DLLs in not an EXE?
- Basic structure of a DLL
- The DLL entry point function
- DllMain() function

```
BOOL WINAPI DllMain(
   HINSTANCE hinstDLL, // handle to DLL module
   DWORD fdwReason,    // reason for calling function
   LPVOID lpvReserved  // reserved
);
```

Dynamic Link Libraries

`DLL_PROCESS_ATTACH`

- Passed to the DLL entry point function when the DLL is being loaded into the virtual address space of the current process as a result of the process starting up or as a result of a call to `LoadLibrary()`

`DLL_THREAD_ATTACH`

- The entry-point function of all DLLs currently attached to the process are called with this value when the current process creates a new thread. Existing threads do not call DLL entry point function with this value

Select proper option in Visual C++ or your compiler to generate a DLL instead of an EXE

DLL exports and imports

- The export table
- How to export and import code (functions) in a DLLs
- Import data
- `__declspec( dllimport ) int i;`
- Export code
- `__declspec( dllexport ) void function(void);`

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

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.

Load-time vs. Runtime Dynamic Linking

- Load-time Dynamic Linking: .LIB file contains all exported function addresses
- Runtime Dynamic Linking: LoadLibrary()
- .DEF module definition files can be used instead of `dllexport`/`dllimport`.
- Using .LIB is safer sometimes in the sense that the programme stub refuses to load the main programme if some DLL can not be loaded.
- .DEF files are less common now, but are more powerful.

Loading a DLL and calling functions in it

```
HMODULE LoadLibrary(
   LPCTSTR lpFileName // file name of module
);

FreeLibrary(hModule);

FARPROC GetProcAddress(
   HMODULE hModule,    // handle to DLL module
   LPCSTR lpProcName   // function name
);
```

Example

Create a DLL **myDll.dll** and its import library **myDll.lib**

```
declspec( dllexport ) int sum(int, int);


int sum(int a, int b)
{
        return a+b;
}
```

Using **myDll.DLL** in your programme

- Specify **myDll.lib** at link-time in the list of import libraries
- Provide a prototype of **int sum(int, int)** in your programme
- Use the function as normal

Example

Using **`myDll.DLL`** in your programme

- Call **`LoadLibrary()`** at runtime to load the DLL
- Call **`GetProcAddress()`** to get a pointer to the function **`sum()`** in your programme
- Use indirection to this pointer to function as normal to call the function.