CS222:
Systems Programming

Memory Management III

February 26th, 2008

Last Class

- Memory Management
  - Overview
  - Heap management
  - Memory-mapped files
  - Dynamic link libraries

Today's Class

- Memory Management
  - Overview
  - Heap management
  - Memory-mapped files
  - Dynamic link libraries

HW1 Grading Delay

- Will be returned ASAP

Free software from Microsoft

- Microsoft is giving away free software to students:
  - Visual Studio 2005 & 2008 Professional
  - Microsoft Expression Studio
  - Windows Server 2003 Standard
  - XNA Game Studio
  - SQL Server 2005 Developer Edition
- https://downloads.channel8.msdn.com/

Efficient Programming

- How can we be more efficient at programming?

Dynamic-Link Libraries

- DLL is a module that contain functions and data that can be used by another module (application or DLL)
- It can define two kinds of functions
  - Exported
    - Intended to be called by other modules
  - Internal
    - Intended to be called only from within the DLL

Dynamic-Link Libraries

- 2 types of dynamic linking
  - Load-time linking (Implicit)
    - Required to link the module with the import library (.lib and .dll files)
    - Easier to use
  - Run-time linking (Explicit)
    - Use `LoadLibrary` or `LoadLibraryEx` to load the DLL at run time
    - Use `GetProcAddress` function to get the addresses for the exported DLL functions

DLL and Memory Management

- A process that loads the DLL
  - Maps it into its own virtual address space
  - Calls the exported DLL functions
- A per-thread reference count for each DLL is maintained
  - When a thread loads the DLL, the count is incremented
  - When the count becomes zero, the DLL is unloaded

DLL and Memory Management

- An exported DLL function runs in the context of thread that calls it
  - The DLL uses the stack of the calling thread and the virtual address space of the calling process
  - The DLL allocates memory from the virtual address space of the calling process

Advantages of Using DLL

- Some advantages over static linking
  - Smaller program image
  - Save system memory, since multiple processes can share a single copy of the DLL
  - Easy to support new versions or alternative implementations
  - Run time decision of which version to use

Load-time Linking (Implicit)

- When the system starts a program, it uses the information the linker placed in the file to locate the names of DLLs
- Search orders (typical)
  - The directory from which the application loaded
  - The system directory
  - The 16-bit system directory
  - The windows directory
  - The current directory
  - The directories that are listed in the PATH

Load-Time Function Interface in DLL

- Use `_declspec (dllexport)` to declare a function to be exportable
  - `_declspec (dllexport) DWORD MyFunction(…);`
- The build process will create .DLL and .LIB files
  - Link the .LIB file with the calling program
- Similar syntax to import a function
  - `_declspec (dllimport) DWORD MyFunction(…);`
- If the calling (importing) client program is written in C++, necessary to specify the C calling convention
  - `extern "C" _declspec (dllimport) DWORD …`

Run-time Linking (Explicit)

- Requires the program to request specifically that a DLL be loaded or freed
- Then, the program obtains the address of the required entry point and uses that address as the pointer in the function call

LoadLibrary

- A function used for mapping the specified executable module into the address space of the calling process
  - Free the library handle using `FreeLibrary`

```
HINSTANCE LoadLibrary(
 LPCTSTR lpLibFileName );
```

GetProcAddress

- A function used for retrieving the address of an exported function or variable from the specified DLL

```
FARPROC GetProcAddress(
 HMODULE hModule,
 LPCTSTR lpProcNameName );
```

Example: myPuts

```
#include <windows.h>
#define EOF (-1)

#ifdef __cplusplus // If used by C++ code,
extern "C" { // we need to export the C interface
#endif

__declspec(dllexport) int myPuts(LPTSTR lpszMsg) {
  DWORD cchWritten;
  HANDLE hStdout;
  BOOL fRet;

  // Get a handle to the standard output device.
  hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
  if (INVALID_HANDLE_VALUE == hStdout) return EOF;

  // Write a null-terminated string to the standard output device.
  while (*lpszMsg != '\0') {
    fRet = WriteFile(hStdout, lpszMsg, 1, &cchWritten, NULL);
    if( (FALSE == fRet) || (1 != cchWritten) ) return EOF;
    lpszMsg++;
  } return 1;
}
#ifdef __cplusplus
} #endif
```

Example: Load-time Linking

```
#include <windows.h>

int myPuts(LPTSTR); // a function from a DLL

int main(VOID)
{
  int Ret = 1;

  Ret = myPuts(TEXT("Message printed using the DLL function\n"));

  return Ret;
}
```

Example: Run-time Linking

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

typedef int (*MYPROC)(LPTSTR);

VOID main(VOID) {
  HINSTANCE hinstLib;
  MYPROC ProcAdd;
  BOOL fFreeResult, fRunTimeLinkSuccess = FALSE;

  // Get a handle to the DLL module.
  hinstLib = LoadLibrary(TEXT("myputs"));

  // If the handle is valid, try to get the function address.
  if (hinstLib != NULL) {
    ProcAdd = (MYPROC) GetProcAddress(hinstLib, TEXT("myPuts"));
    if (NULL != ProcAdd) {
      fRunTimeLinkSuccess = TRUE;
      (ProcAdd) (TEXT("Message via DLL function\n"));
    } // Free the DLL module.

    fFreeResult = FreeLibrary(hinstLib);
  } // If unable to call the DLL function, use an alternative.
  if (! fRunTimeLinkSuccess) printf("Message via alternative method\n");
}
```

The DLL Entry Point

- Optional
- Invoked with a process attaches or detaches the DLL
- Serialized by the system
  - Don't use blocking calls

```
BOOL DllMain(
 HINSTANCE hDll,
 DWORD Reason,
 LPVOID Reserved);
```

DllMain

```
BOOL DllMain(
 HINSTANCE hDll,
 DWORD Reason,
 LPVOID Reserved);
```

- hDll
  - Instance obtained from `LoadLibrary`
- Reserved
  - If NULL, process attachment was caused by LoadLibrary. Otherwise, implicit load-time linking was used

DllMain

```
BOOL DllMain(
 HINSTANCE hDll,
 DWORD Reason,
 LPVOID Reserved);
```

- Reason (one of four values)
  - DLL_PROCESS_ATTACH
  - DLL_THREAD_ATTACH
  - DLL_THREAD_DETACH
  - DLL_PROCESS_DETACH

Quiz

- There will be a Quiz in Tuesday's class
  - Covering everything we've seen so far…

Homework 3

- Writing a DLL
  - Writing programs to use your DLL
    - Using load time linking
    - Using run time linking

- Due Tuesday, March 4th

Midterm

- Thursday, March 6th
- We will review briefly in class on March 4th.
- Last year's midterm will be available on the class website

Review

- Memory management
  - Overview
  - Heap management
  - Memory-mapped files
  - Dynamic link libraries

- Recommended reading for next class
  - Chapter 6 in Windows System Programming