CS222: Systems Programming

Memory Management II

February 21st, 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

Heap Review

- Memory reserved by HeapCreate is *not necessarily contiguous*
- Memory allocated by HeapAlloc *is contiguous*

GetProcessHeap

- A function used for obtaining a handle to the heap of the calling process
  - Heap handle is necessary when you are allocating memory
  - Each process has its own default heap, which is used by `malloc`

```
HANDLE GetProcessHeap( VOID );
```

Return: The handle for the process's heap: NULL on failure

HeapCreate

- A function used for creating a heap object that can be used by the calling process
  - Reserve space in the virtual address space of the process
  - Allocate physical storage for a specified initial portion
  - `flOptions`
    - `HEAP_GENERATE_EXCEPTIONS`
    - `HEAP_NO_SERIALIZE`

```
HANDLE HeapCreate(
 DWORD flOptions,
 SIZE_T dwInitialSize,
 SIZE_T dwMaximumSize);
```

Return: The handle for the heap: NULL on failure

HeapDestroy

- A function used for destroying an entire heap
  - Decommit and release all the pages of a private heap object
  - Be careful not to destroy the process's heap
- Destroying a heap is a quick way to free data structures without traversing them to delete one element at a time

```
BOOL HeapDestroy( HANDLE hHeap );
```

HeapAlloc

- A function used for allocating a block of memory from a heap
  - `dwFlags`
    - `HEAP_GENERATE_EXCEPTIONS`
    - `HEAP_NO_SERIALIZE`
    - `HEAP_ZERO_MEMORY`
- Use `HeapFree` function to deallocate memory

```
LPVOID HeapAlloc(
 HANDLE hHeap,
 DWORD dwFlags,
 SIZE_T dwBytes);
```

Return: A pointer to the allocated memory block, or NULL on failure

HeapReAlloc

- A function used for reallocating a block of memory from a heap

```
LPVOID HeapReAlloc(
 HANDLE hHeap,
 DWORD dwFlags,
 LPVOID lpMem
 SIZE_T dwBytes);
```

Return: A pointer to the reallocated memory block, or NULL on failure

HEAP_NO_SERIALIZE

- Use for small performance gain
- Requirements
  - No multi-threaded programming

    or
  - Each thread uses its own heap

    or
  - Program has its own mutual exclusion mechanism

Summary: Heap Management

- The normal process for using heaps is as follows
  1. Get a heap handle with either `HeapCreate` or `GetProcessHeap`
  2. Allocate blocks within the heap using `HeapAlloc`
  3. Optionally, free some or all of the individual blocks with `HeapFree`
  4. Destroy the heap and close the handle with `HeapDestroy`

Memory-mapped Files

- Memory-mapped file functionality
  - Map virtual memory space directly to normal files
- Advantages
  - No need to perform direct file I/O
  - Data structures created in memory will be saved in the file for later use
  - In-memory algorithms can process file data even though the file is much larger than available physical memory
  - Improvement of file processing performance
  - No need to manage buffers and the file data
  - Multiple processes can share memory

File Mapping Objects

- As the first step to use MMF, we
  - Need to create a file mapping object on an open file
  - Can give names to the object so that it can be accessible to other processes for shared memory
  - Can protect the object using security attributes
- Use `CreateFileMapping` function

CreateFileMapping

- A function used for creating or opening a named or unnamed file mapping object for specified file

```
HANDLE CreateFileMapping (
 HANDLE hFile,
 LPSECURITY_ATTRIBUTE lpsa,
 DWORD dwProtect,
 DWORD dwMaximumSizeHigh,
 DWORD dwMaximumSizeLow,
 LPCTSTR lpMapName
);

Return: If function succeeds, the return value is a handle.
        Otherwise, the return value is NULL
```

Example: CreateFileMapping

```
hMap = CreateFileMapping(...);

if (hMap != NULL && GetLastError() == ERROR_ALREADY_EXISTS)
{
  CloseHandle(hMap);
  hMap = NULL;
}

return hMap;
```

OpenFileMapping

- A function used for opening a named file mapping object

```
HANDLE OpenFileMapping (
 DWORD dwDesiredAccess,
 BOOL hInheritHandle,
 LPCTSTR lpName
);

Return: If function succeeds, the return value is a handle.
        Otherwise, the return value is NULL
```

Mapping Address to Object

- As the second step, we
  - Need to allocate virtual memory space and map it to a file through the mapping object
  - Similar to `HeapAlloc`
    - Much coarser – larger allocation units
- Use `MapViewOfFile` function

MapViewOfFile

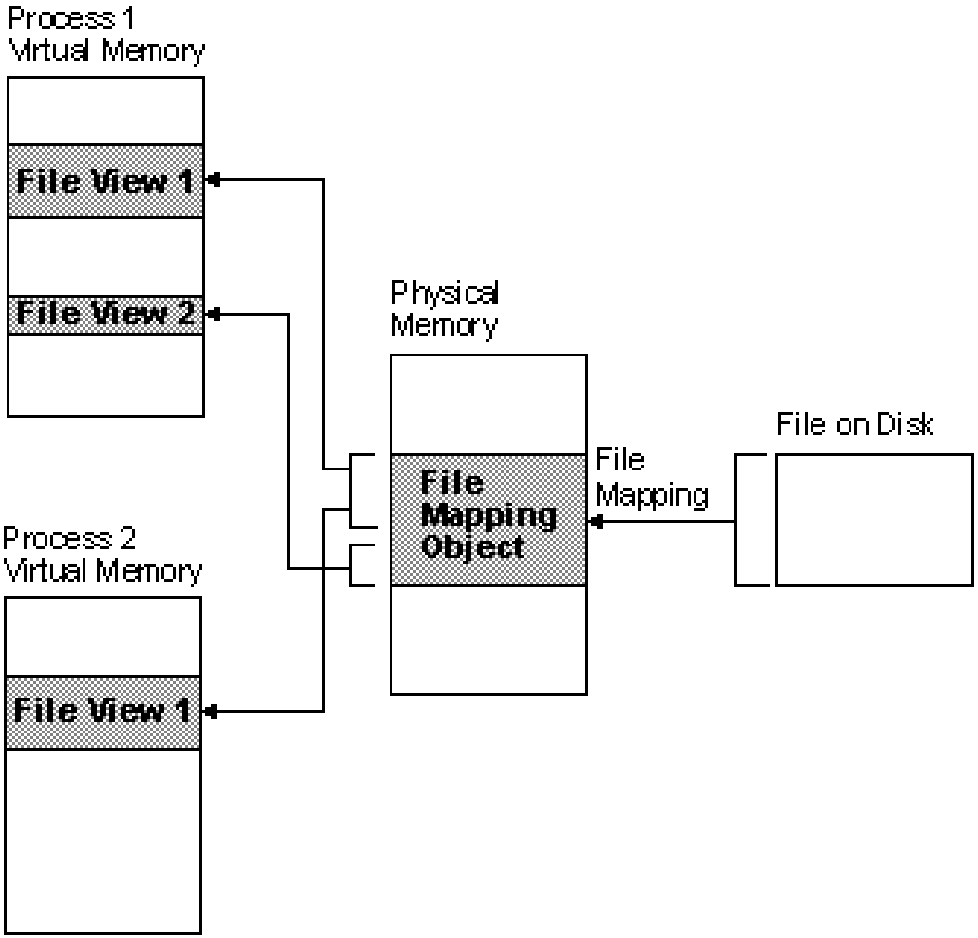
- A function used for mapping a view of a file mapping into the address space of a calling process

```
LPVOID MapViewOffile (
 HANDLE hFileMappingOject,
 DWORD dwDisiredAccess,
 DWORD dwFileOffsetHigh,
 DWORD dwFileOffsetLow,
 SIZE_T dwNumberOfBytesToMap
);

Return: If function succeeds, the return value is starting
        address of the mapped view. Otherwise, NULL
```

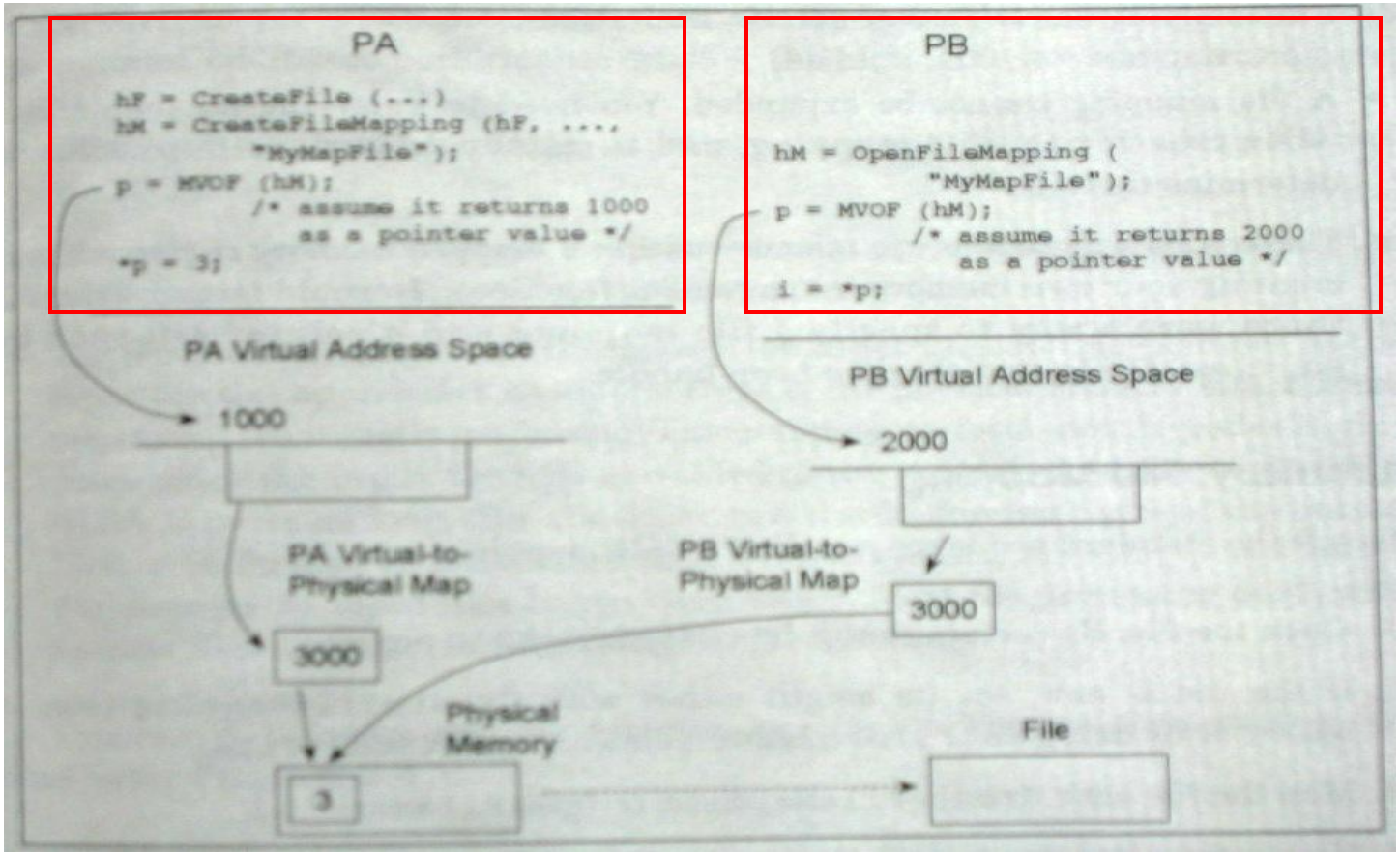
MapViewOfFile

- `MapViewOfFileEx` is similar
  - Must specify a starting memory address
- Use `UnmapViewOfFile` to release memory

Addresses Mapped to a File



Shared Memory



FlushViewOfFile

- Forces system to write changed pages to disk
  - Can be troublesome when concurrently using regular I/O
    - ReadFile
    - WriteFile
  - Coherency is not ensured
  - Do not use these I/O functions for mapped files

Example: MMF-P1

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

#define BUF_SIZE 256
TCHAR szName[]=TEXT("MyFileMappingObject");
TCHAR szMsg[]=TEXT("Message from first process");

void main(){
   HANDLE hMapFile;
   LPCTSTR pBuf;

   hMapFile = CreateFileMapping(
                 INVALID_HANDLE_VALUE,    // use paging file
                 NULL,                    // default security
                 PAGE_READWRITE,          // read/write access
                 0,                       // max. object size
                 BUF_SIZE,                // buffer size
                 szName);                 // name of mapping object
   if (hMapFile == NULL || hMapFile == INVALID_HANDLE_VALUE) {
      printf("Could not create file mapping object (%d).\n",
             GetLastError());
      return;
   }
```

Example: MMF-P1

```
 pBuf = (LPTSTR) MapViewOfFile(hMapFile,   // handle to map object
                         FILE_MAP_ALL_ACCESS, // read/write permission
                         0,
                         0,
                         BUF_SIZE);

   if (pBuf == NULL)
   {
      printf("Could not map view of file (%d).\n",
             GetLastError());
      return;
   }


   CopyMemory((PVOID)pBuf, szMsg, strlen(szMsg));
   getch();

   UnmapViewOfFile(pBuf);

   CloseHandle(hMapFile);
}
```

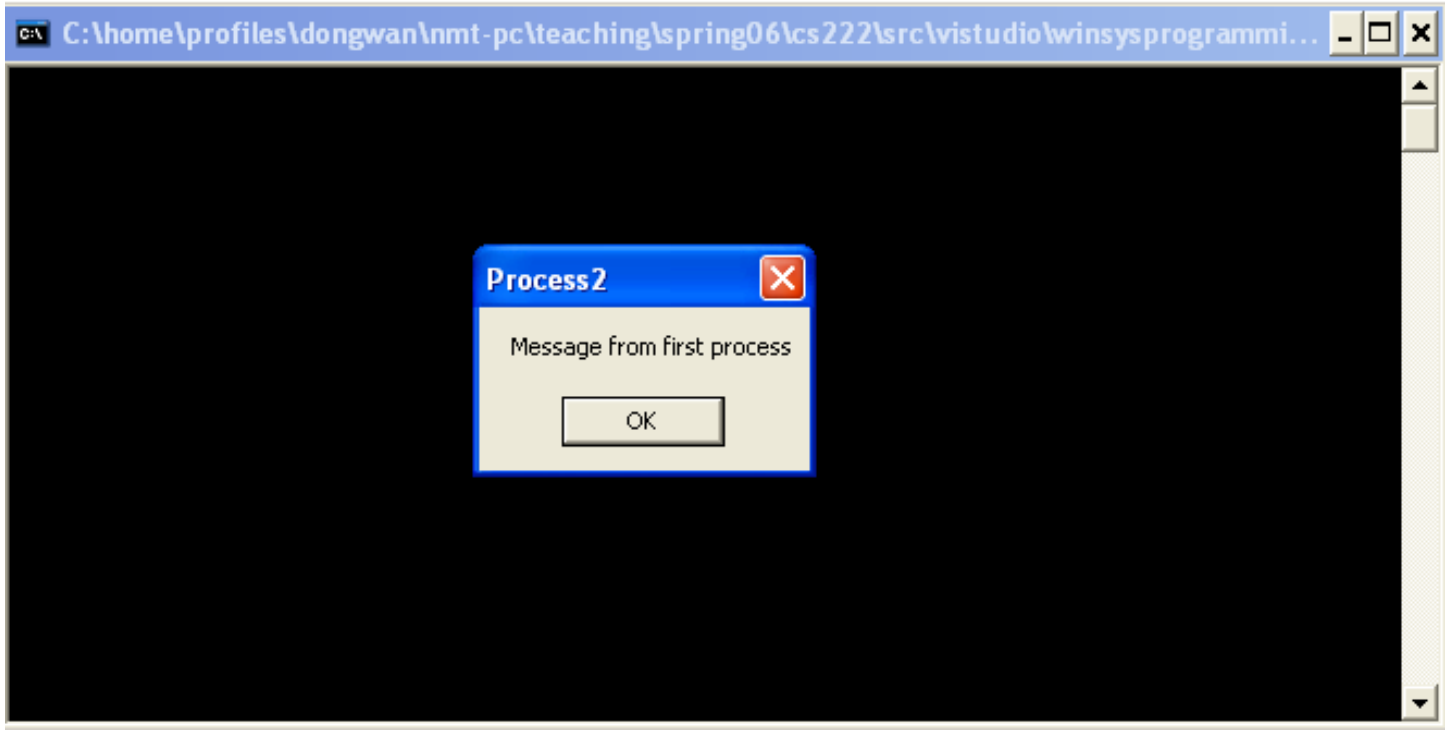
Example: MMF-P2

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

#define BUF_SIZE 256
TCHAR szName[]=TEXT("MyFileMappingObject");

void main()
{
   HANDLE hMapFile;
   LPCTSTR pBuf;

   hMapFile = OpenFileMapping(
                  FILE_MAP_ALL_ACCESS,   // read/write access
                  FALSE,                 // do not inherit the name
                  szName);               // name of mapping object

   if (hMapFile == NULL)
   {
      printf("Could not open file mapping object (%d).\n",
             GetLastError());
      return;
   }
```

Example: MMF-P2

```
pBuf = MapViewOfFile(hMapFile,    // handle to mapping object
              FILE_MAP_ALL_ACCESS,  // read/write permission
              0,
              0,
              BUF_SIZE);

   if (pBuf == NULL)
   {
      printf("Could not map view of file (%d).\n",
             GetLastError());
      return;
   }

 MessageBox(NULL, pBuf, TEXT("Process2"), MB_OK);

   UnmapViewOfFile(pBuf);

   CloseHandle(hMapFile);
}
```

Result


C:\home\profiles\dongwan\nmt-pc\teaching\spring06\cs222\src\vistudio\winsysprogrammi...
Process2
Message from first process
OK

File Mapping Limitation

- File mapping is powerful and useful, however
  - File mapping cannot be expanded
  - No way to allocate memory within a mapped memory region

Summary: MMF

- Standard sequence required to use MMF
  1. Open the file
  2. If the file is new, set the file length either with `CreateFileMapping` or by using `SetFilePoiner` followed by `SetEndOfFile`
  3. Map the file with `CreateFileMapping` or `OpenFileMapping`
  4. Create one or more views with `MapViewOfFile`
  5. Access the file through memory references
  6. On completion, un-map the file and close handles for file mapping object as well as file

Review

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

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