CS222: Systems Programming

Memory Management

February 19th, 2008

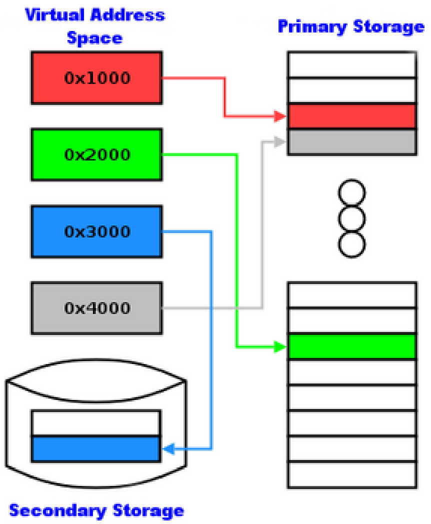
Last Class

- Error Handling
  - Exception Handling
  - Console Control Handlers
  - Vectored Exception Handling

Today's Class

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

Memory Management I



Process and Memory Space

- Each process has its own virtual address space
  - Up to 4 GB of memory (32-bit)
    - Actually 2GB (3GB possible)
- All threads of a process can access its virtual address space
  - However, they cannot access memory that belongs to another process

Virtual Address Space

- Virtual address of a process does not represent the actual physical location of an object in memory
- Each process maintains its page map
  - Internal data structure used to translate virtual addresses into corresponding physical addresses
  - Each time a thread references an address, the system translates the virtual address to physical address

Virtual and Physical Memory

- Virtual address space of a process can be smaller or larger than the total physical memory available on the computer
- The subset of the virtual address space of a process that resides in physical memory is called working set
  - If the threads of a process attempt to use more physical memory than is currently available, then the system pages some memory contents to disk

Pages

- A page is a unit of memory, into which physical storage and the virtual address space of each process are organized
  - Size depends on the host computer
- When a page is moved in physical memory, the system updates the page maps of the affected processes
- When the system needs space in physical memory, it moves the least recently used pages of physical memory to the paging file

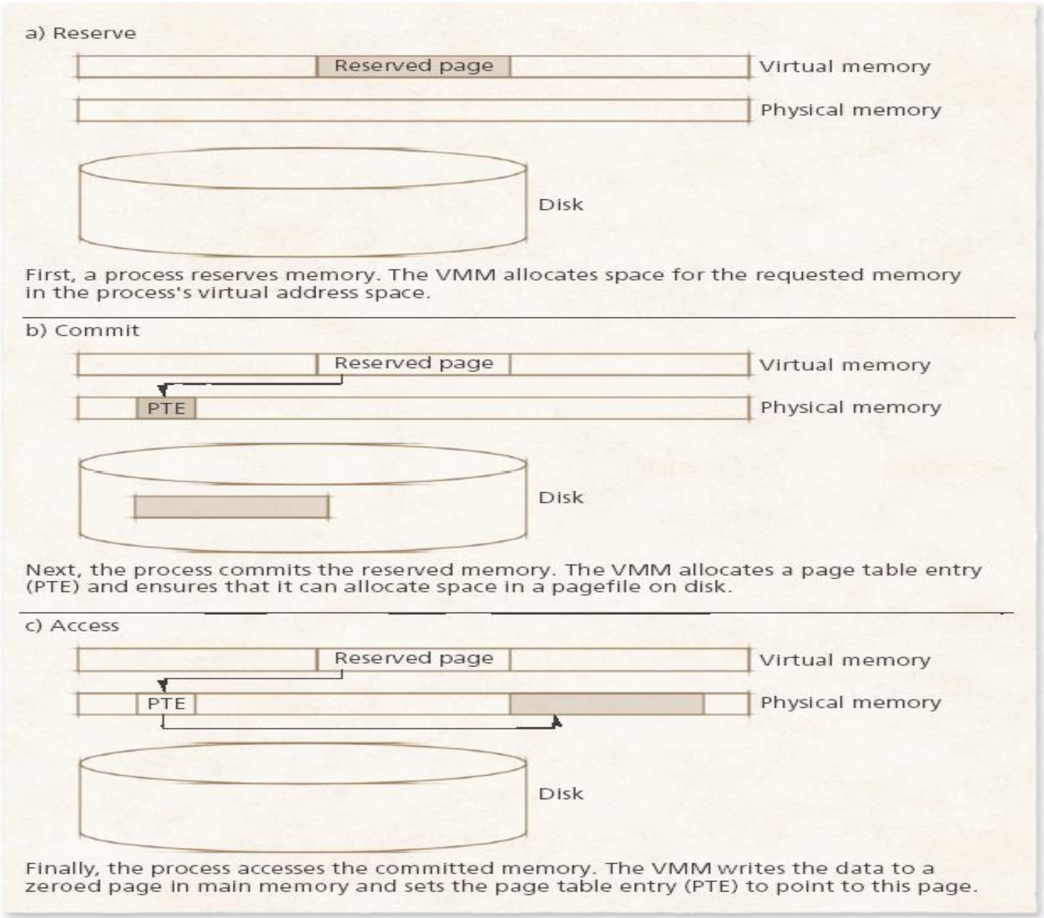
Page State

- The pages of a process's virtual address space can be in one of the following states
  - Free
    - Neither committed nor reserved, but available
    - Not accessible to the process
    - Attempting to read from or write to a free page results in access violation exception
    - `VirtualFree` or `VirtualFreeEx`

Page State, cont

- Reserved
  - Reserved for future use
  - Address range cannot be used by other allocation functions
  - Not accessible and has no physical storage associated with it
  - Available to be committed
  - `VirtualAlloc` or `VirtualAllocEx`
- Committed
  - Physical storage is allocated, and access is controlled
  - When process terminates, it is released
  - `VirtualAlloc` or `VirtualAllocEx`

Page State, cont



Scope of Allocated Memory

- All memory allocated by memory allocation functions
  - Is process-wide
  - `HeapAlloc`, `VirtualAlloc`, `GlobalAlloc`, `LocalAlloc`
- All memory allocated by a DLL is allocated in the address space of the process that called the DLL
- In order to create shared memory, we must use file mapping

Page Faults

- References to pages not in memory
  - Most virtual pages will not be in physical memory
  - OS loads the data from disk, either from
    - System swap file, or
    - Normal file
- For performance purpose, programs should be designed minimize page faults

GetSystemInfo

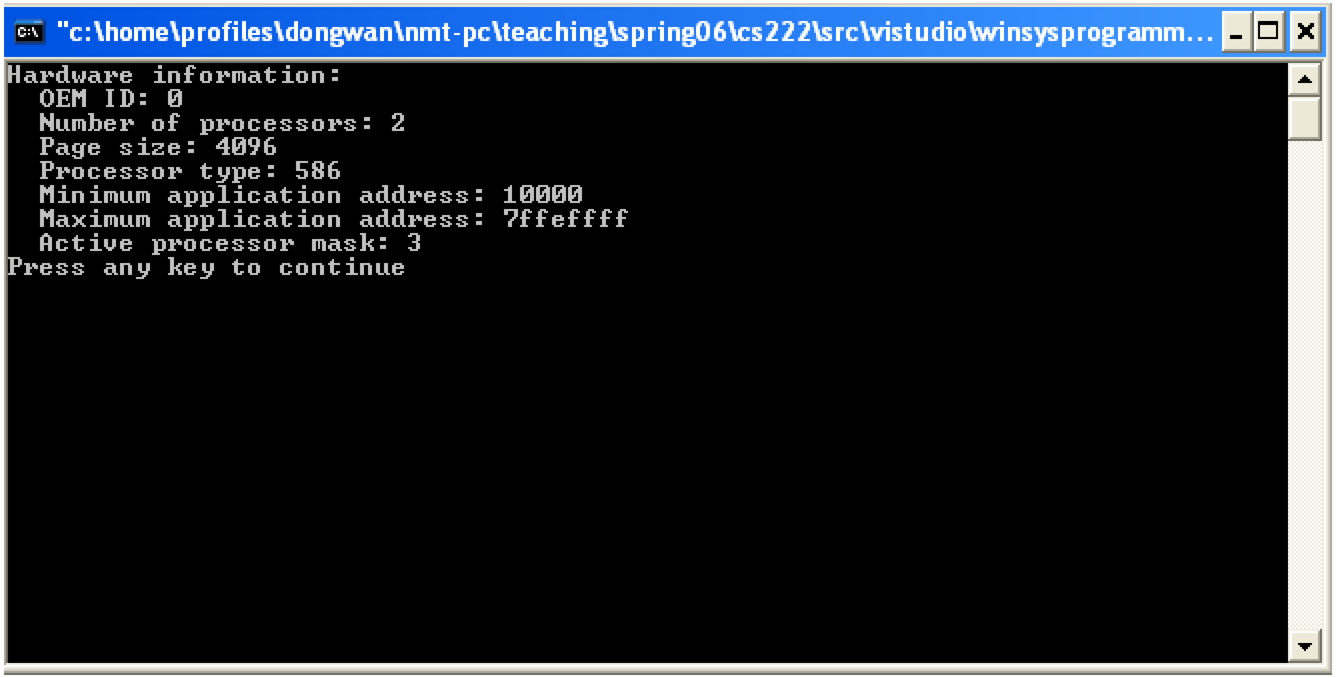
- A function returning information about the current system
  - `SYSTEM_INFO` structure contains information including the architecture and type of a processor, the number of processors, page size, etc

```
VOID GetSystemInfo(
 LPSYSTEM_INFO lpSystemInf);
```

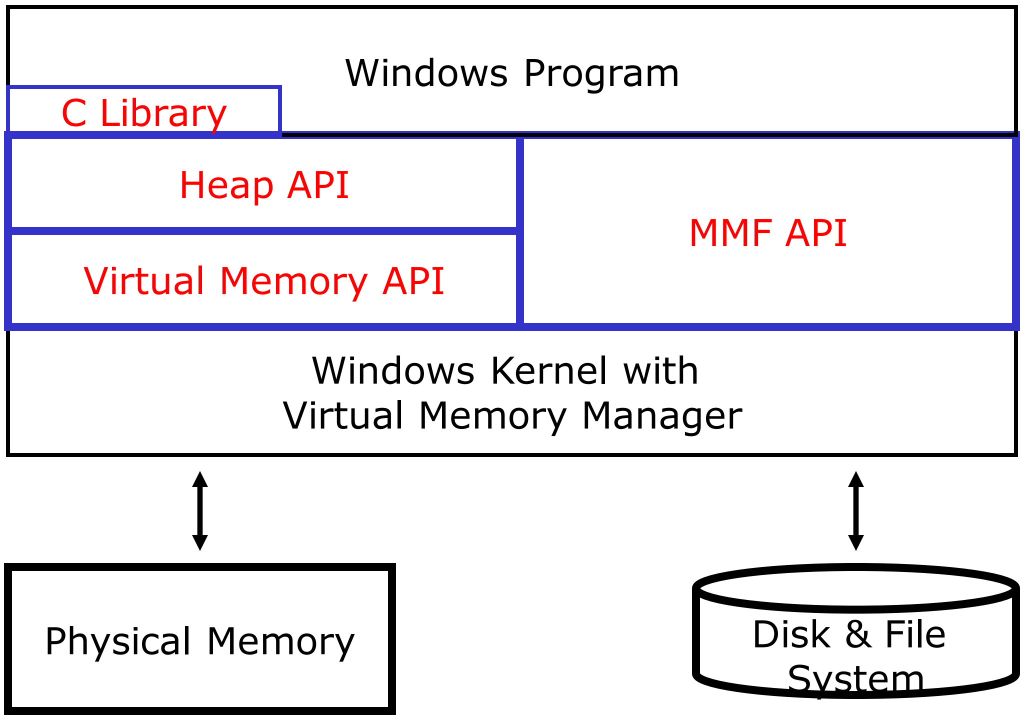
Example: GetSystemInfo

```
void main()
{
   SYSTEM_INFO siSysInfo;

   GetSystemInfo(&siSysInfo);

   // Display the contents of the SYSTEM_INFO structure.

   printf("Hardware information: \n");
   printf("  OEM ID: %u\n", siSysInfo.dwOemId);
   printf("  Number of processors: %u\n",
      siSysInfo.dwNumberOfProcessors);
   printf("  Page size: %u\n", siSysInfo.dwPageSize);
   printf("  Processor type: %u\n", siSysInfo.dwProcessorType);
   printf("  Minimum application address: %lx\n",
      siSysInfo.lpMinimumApplicationAddress);
   printf("  Maximum application address: %lx\n",
      siSysInfo.lpMaximumApplicationAddress);
   printf("  Active processor mask: %u\n",
      siSysInfo.dwActiveProcessorMask);
}
```

Example: GetSystemInfo



Windows Memory Management



Heaps

- A heap is used for allocating and freeing objects dynamically for use by the program. Heap operations are called for when
  - The number and size of objects needed by the program are not known ahead of time
  - An object is too large to fit into a stack allocator

Heap Management

- A process can contain several heaps for following reasons
  - Fairness
  - Multithreaded performance
  - Allocation efficiency
  - Deallocation efficiency
  - Locality of reference efficiency
- Often a single heap is sufficient. In that case, <u>use the C library memory management functions</u>
  - `malloc`, `calloc`, `realloc`, `free`, etc

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 <u>creating a heap object</u> 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 date 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`

Example: HeapCreate/HeapAlloc

```
HANDLE hHeap;
SIZE_T nBufferSize;

...

/* allocate memory for the buffer */
__try{
  hHeap = HeapCreate(HEAP_GENERATE_EXCEPTIONS | HEAP_NO_SERIALIZE,
                          nBufferSize, 0);    // growable heap size

  cBuffer = HeapAlloc(hHeap, HEAP_ZERO_MEMORY, sizeof(TCHAR)*nBufferSize);
}

__except(EXCEPTION_EXECUTE_HANDLER){
  printf("Exception occurred... : %x", GetExceptionCode());
}

...

/* free allocated memory */
HeapDestroy(hHeap);
```

Review

- Memory management
  - Overview
  - Heap management
  - Memory-mapped files
  - Dynamic link libraries
- Recommended reading for next class
  - Chapter 6 in Windows System Programming

Next Class

- Quiz
- Homework due next Tuesday