**CS149: Spring 2016**

**Project #1: Virtual Memory Management Simulation (VMMS) - 200 Points**

**Due Date: April 10, 2016**

**Description:**

You are responsible to write a simulated memory manager on the Windows platform. This package will contain several components. That includes a DLL which contains all the memory service functions (API). It contains an executable console to initialize the physical memory and virtual memory startup values. The console takes commands to dump the current content of both physical and virtual memory. It contains a header file and an import library for other programs to use the memory functions. In addition, you will create a set of functional test cases embedded in other executables to test all the code paths. You will hand in all the parts in a single zip file on Canvas.

This VMMS is quite different from a real life memory manager. Upon allocation request from client application, VMMS will “reserve” a block of memory from the virtual memory file “VMMS.MEM”. For any read/write requests, it must be performed through the VMMS physical memory pool. Therefore the physical memory acted as our “cache” area for any read/write requests to a memory mapped file.

**Project Specification:**

**Virtual Memory Management Shared Library: vmms.dll**

char\* vmms\_malloc ( int size, int\* error\_code )

- Allocate a piece of memory given the input size.

- If successful, returns a valid pointer. Otherwise it returns NULL (0) and set the error\_code.

- Possible errors: 100

int vmms\_memset ( char\* dest\_ptr, char c, int size )

- Set the destination buffer with a character of certain size.

- If successful, returns 0. Otherwise it returns an error code.

- Possible errors: 101, 102

int vmms\_memcpy ( char\* dest\_ptr, char\* src\_ptr, int size )

- Copy the fixed number of bytes from source to destination.

- If successful, returns 0. Otherwise it returns an error code.

- Possible errors: 101, 103

- Must allow external buffer to be pass in as dest\_ptr. (read only request)

int vmms\_print ( char\* src\_ptr, int size )

- Print the number of characters to STDOUT.

- If size=0, then print until the first hex 0 to STDOUT.

- If successful, returns 0. Otherwise it returns an error code.

- Possible errors: 103

- Must allow external buffer to be pass in as src\_ptr. (read only request)

int vmms\_free ( char\* mem\_ptr )

- Free the allocated memory.

- If successful, returns 0. Otherwise it returns an error code.

- Possible errors: 104

**Memory Management Console: mmc.exe (Usage: mmc physical boundary)**

- “physical” is the simulated physical memory size

- “boundary” is the minimum number of bytes in an allocated memory region

**MMC functions:**

- “D” [ filename ] - dumps out the memory to the screen. If filename is specified, also save to binary file.

- “M” [ filename ] - displays the content of the internal memory mapping table in a readable format.

- “E” - exit the console and free up the “shared” memory.

**Memory Management Interfaces: vmms.h and vmms.lib**

- vmms.h contains the external functional prototypes and error constants used in client program.

- vmms.lib is required for the client program at compile and link time.

**Memory Management Error Codes:**

100 OUT\_OF\_MEM Out of memory.

101 MEM\_TOO\_SMALL Destination memory buffer is too small for this operation.

102 INVALID\_DEST\_ADDR Invalid destination memory address.

103 INVALID\_CPY\_ADDR Invalid destination & source memory address.

104 INVALID\_MEM\_ADDR Invalid memory address. (includes freeing the same pointer twice)

**Implementation Specification:**

**Minimum set of memory mapping table information:**

PID = client process ID

Request Size = total size of requested memory

Actual Size = actual memory size including boundary size

Client Address = address returns to client’s malloc() request

Last Reference = time of the last write request (memset or memcpy)

Head of free list data structure = a list of memory free list locations and sizes

**Rules & restrictions that should make your life easier:**

Assume the memory requests are serialized from different processes, thus no lock/unlock is necessary.

At the end of each memory write call including compaction, flush to disk (vmms.mem) immediately.

At the end of each free() call, mark the entries in page table.

Do not cleanup any memory location inside the physical memory explicitly.

Use PID 0 entry to store the last free list entry.

Note that the same process can request memory multiple times (including recursion).

You must not hold a file lock on the “**VMMS.MEM**” file after each memory request.

For each memory request, you must log the request along with timestamp to file “**VMMS.LOG**”.

**Useful System Calls:**

GetSystemTime()

\_getpid()

**MMS.LOG content:**

<timestamp> <program name> <pid> <operation>

<timestamp> is a fixed 14 characters string with the following format, yyyymmddhhmmss

<program name> is the client program exe name

<pid> is the process ID of the client program

<operation> is full name of the function “mms\_\*” following by the return values 4 bytes & its parameters

e.g. 20060214211000 test1.exe 2001 mms\_memset 0 4096 65 256

**Memory dump content:**

Physical:

<address> <bytes> <character representation>

...

Virtual:

<address> <bytes> <character representation>

...

<address> is a 4 byte address display in hex

<bytes> are 16 bytes display in hex, each byte separated by a blank

<char rep> are a set of printable character in this line.