Windows Memory Management

Aastha Trehan, Ritika Grover, Prateek Puri
Dronacharya College Of Engineering, Gurgaon

Abstract- **Memory plays a key part in any kind of processing that takes place in a computer. Virtual memory is critical for doing Windows memory management. There are several ways a virtual memory can be allocated. Also there are various functions that be used for doing the effective memory management.**

Index Tersms- **Virtual memory, Memory allocations, Memory functions, Memory pages**

I. Introduction

The memory manager implements virtual memory provides a core set of services such as memory mapped files, copy-on-write memory, large memory support, and underlying support for the cache manager. Each process on 32-bit Microsoft Windows has its own virtual address space that enables addressing up to 4 gigabytes of memory. Each process on 64-bit Windows has a virtual address space of 8 terabytes. All threads of a process can access its virtual address space. However, threads cannot access memory that belongs to another process, which protects a process from being corrupted by another process.

II. Virtual Memory Functions

The virtual memory functions enable a process to manipulate or determine the status of pages in its virtual address space. They can perform the following operations:

- Reserve a range of a process's virtual address space. Reserving address space does not allocate any physical storage, but it prevents other allocation operations from using the specified range. It does not affect the virtual address spaces of other processes. Reserving pages prevents needless consumption of physical storage, while enabling a process to reserve a range of its address space into which a dynamic data structure can grow. The process can allocate physical storage for this space, as needed.
- Commit a range of reserved pages in a process's virtual address space so that physical storage (either in RAM or on disk) is accessible only to the allocating process.
- Specify read/write, read-only, or no access for a range of committed pages. This differs from the standard allocation functions that always allocate pages with read/write access.
- Free a range of reserved pages, making the range of virtual addresses available for subsequent allocation operations by the calling process.
- Decommit a range of committed pages, releasing their physical storage and making it available for subsequent allocation by any process.
- Lock one or more pages of committed memory into physical memory (RAM) so that the system cannot swap the pages out to the paging file.
- Obtain information about a range of pages in the virtual address space of the calling process or a specified process.
- Change the access protection for a specified range of committed pages in the virtual address space of the calling process or a specified process.

III. Allocating Virtual Memory

The virtual memory functions manipulate pages of memory. The functions use the size of a page on the current computer to round off specified sizes and addresses. The **VirtualAlloc** function performs one of the following operations:

- Reserves one or more free pages.

- Commits one or more reserved pages.
- Reserves and commits one or more free pages.

You can specify the starting address of the pages to be reserved or committed, or you can allow the system to determine the address. The function rounds the specified address to the appropriate page boundary. Reserved pages are not accessible, but committed pages can be allocated with **PAGE_READWRITE**, **PAGE_READONLY**, or **PAGE_NOACCESS** access. When pages are committed, memory charges are allocated from the overall size of RAM and paging files on disk, but each page is initialized and loaded into physical memory only at the first attempt to read from or write to that page. You can use normal pointer references to access memory committed by the **VirtualAlloc** function.

IV. COMPARING MEMORY ALLOCATION METHODS

The following is a brief comparison of the various memory allocation methods:

- **CoTaskMemAlloc**
- **GlobalAlloc**
- **HeapAlloc**
- **LocalAlloc**
- **malloc**
- **new**
- **VirtualAlloc**

Although the **GlobalAlloc**, **LocalAlloc**, and **HeapAlloc** functions ultimately allocate memory from the same heap, each provides a slightly different set of functionality. For example, **HeapAlloc** can be instructed to raise an exception if memory could not be allocated, a capability not available with **LocalAlloc**. **LocalAlloc** supports allocation of handles which permit the underlying memory to be moved by a reallocation without changing the handle value, a capability not available with **HeapAlloc**.
Starting with 32-bit Windows, **GlobalAlloc** and **LocalAlloc** are implemented as wrapper functions that call **HeapAlloc**using a handle to the process's default heap. Therefore, **GlobalAlloc** and **LocalAlloc** have greater overhead than**HeapAlloc**.
Because the different heap allocators provide distinctive functionality by using different mechanisms, you must free memory with the correct function. For example, memory allocated with **HeapAlloc** must be freed with **HeapFree** and not **LocalFree** or **GlobalFree**. Memory allocated with **GlobalAlloc** or **LocalAlloc** must be queried, validated, and released with the corresponding global or local function.
The **VirtualAlloc** function allows you to specify additional options for memory allocation. However, its allocations use a page granularity, so using **VirtualAlloc** can result in higher memory usage.
The **malloc** function has the disadvantage of being run-time dependent. The **new** operator has the disadvantage of being compiler dependent and language dependent.
The **CoTaskMemAlloc** function has the advantage of working well in either C, C++, or Visual Basic. It is also the only way to share memory in a COM-based application, since MIDL uses **CoTaskMemAlloc** and **CoTaskMemFree** to marshal memory.

V. FREEING VIRTUAL MEMORY

The **VirtualFree** function decommits and releases pages according to the following rules:

- Decommits one or more committed pages, changing the state of the pages to reserved. Decommitting pages releases the physical storage associated with the pages, making it available to be allocated by any process. Any block of committed pages can be decommitted.
- Releases a block of one or more reserved pages, changing the state of the pages to free. Releasing a block of pages makes the range of reserved addresses available to be allocated by the process. Reserved pages can be released only by freeing the entire block that was initially reserved by **VirtualAlloc**.
- Decommits and releases a block of one or more committed pages simultaneously, changing the state of the pages to free. The specified block must include the entire block initially reserved by **VirtualAlloc**, and all of the pages must be currently committed.

After a memory block is released or decommitted, you can never refer to it again. Any information that may have been in that memory is gone forever. Attempting to read from or write to a free page results in an access violation exception. If you require information, do not decommit or free memory containing that information.
To specify that the data in a memory range is no longer of interest, call **VirtualAlloc** with **MEM_RESET**. The pages will not be read from or written to the paging file. However, the memory block can be used again later.

VI. Working With Pages

To determine the size of a page on the current computer, use the **GetSystemInfo** function. The **VirtualQuery** and **VirtualQueryEx** functions return information about a region of consecutive pages beginning at a specified address in the address space of a process. **VirtualQuery** returns information about memory in the calling process. **VirtualQueryEx** returns information about memory in a specified process and is used to support debuggers that need information about a process being debugged. The region of pages is bounded by the specified address rounded down to the nearest page boundary. It extends through all subsequent pages with the following attributes in common:

- The state of all pages is the same: either committed, reserved, or free.
- If the initial page is not free, all pages in the region are part of the same initial allocation of pages that were reserved by a call to **VirtualAlloc**.
- The access protection of all pages is the same (that is, **PAGE_READONLY**, **PAGE_READWRITE**, or **PAGE_NOACCESS**).

The **VirtualLock** function enables a process to lock one or more pages of committed memory into physical memory (RAM), preventing the system from swapping the pages out to the paging file. It can be used to ensure that critical data is accessible without disk access. Locking pages into memory is dangerous because it restricts the system's ability to manage memory. Excessive use of **VirtualLock** can degrade system performance by causing executable code to be swapped out to the paging file. The **VirtualUnlock** function unlocks memory locked by **VirtualLock**. The **VirtualProtect** function enables a process to modify the access protection of any committed page in the address space of a process. For example, a process can allocate read/write pages to store sensitive data, and then it can change the access to read only or no access to protect against accidental overwriting. **VirtualProtect** is typically used with pages allocated by **VirtualAlloc**, but it also works with pages committed by any of the other allocation functions. However, **VirtualProtect** changes the protection of entire pages, and pointers returned by the other functions are not necessarily aligned on page boundaries. The **VirtualProtectEx** function is similar to **VirtualProtect**, except it changes the protection of memory in a specified process. Changing the protection is useful to debuggers in accessing the memory of a process being debugged.

VII. Memory Management Functions

A. *General Memory Functions*

This topic describes the memory management functions:

The following functions are used in memory management.

Function	Description
AddSecureMemoryCacheCallback	Registers a callback function to be called when a secured memory range is freed or its protections are changed.
CopyMemory	Copies a block of memory from one location to another.
CreateMemoryResourceNotification	Creates a memory resource notification object.
FillMemory	Fills a block of memory with a specified value.
GetLargePageMinimum	Retrieves the minimum size of a large page.
GetPhysicallyInstalledSystemMemory	Retrieves the amount of RAM that is physically installed on the computer.
GetSystemFileCacheSize	Retrieves the current size limits for the working set of the system cache.
GetWriteWatch	Retrieves the addresses of the pages that have been written to in a region of virtual memory.
GlobalMemoryStatusEx	Obtains information about the system's current usage of both physical and virtual memory.
MoveMemory	Moves a block of memory from one location to another.
QueryMemoryResourceNotification	Retrieves the state of the specified memory resource object.
RemoveSecureMemoryCacheCallback	Unregisters a callback function that was previously registered with the**AddSecureMemoryCacheCallback** function.
ResetWriteWatch	Resets the write-tracking state for a region of virtual memory.
SecureMemoryCacheCallback	An application-defined function that is called when a secured memory range is freed or its protections are changed.
SecureZeroMemory	Fills a block of memory with zeros.
SetSystemFileCacheSize	Limits the size of the working set for the file system cache.

Function	Description
ZeroMemory	Fills a block of memory with zeros.

B. Data Execution Prevention Functions

The following functions are used with Data Execution Prevention (DEP).

Function	Description
GetProcessDEPPolicy	Retrieves DEP settings for a process.
GetSystemDEPPolicy	Retrieves DEP settings for the system.
SetProcessDEPPolicy	Changes DEP settings for a process.

C. File Mapping Functions

The following functions are used in file mapping.

Function	Description
CreateFileMapping	Creates or opens a named or unnamed file mapping object for a specified file.
CreateFileMappingFromApp	Creates or opens a named or unnamed file mapping object for a specified file from a Windows Store app.
CreateFileMappingNuma	Creates or opens a named or unnamed file mapping object for a specified file, and specifies the NUMA node for the physical memory.
FlushViewOfFile	Writes to the disk a byte range within a mapped view of a file.
GetMappedFileName	Checks whether the specified address is within a memory-mapped file in the address space of the specified process. If so, the function returns the name of the memory-mapped file.
MapViewOfFile	Maps a view of a file mapping into the address space of a calling process.
MapViewOfFileEx	Maps a view of a file mapping into the address space of a calling process. A caller can optionally specify a suggested memory

	address for the view.
MapViewOfFileExNuma	Maps a view of a file mapping into the address space of a calling process, and specifies the NUMA node for the physical memory.
MapViewOfFileFromApp	Maps a view of a file mapping into the address space of a calling process from a Windows Store app.
OpenFileMapping	Opens a named file mapping object.
UnmapViewOfFile	Unmaps a mapped view of a file from the calling process's address space.

D. AWE Functions

The following are the AWE functions.

Function	**Description**
AllocateUserPhysicalPages	Allocates physical memory pages to be mapped and unmapped within any AWE region of the process.
FreeUserPhysicalPages	Frees physical memory pages previously allocated with**AllocateUserPhysicalPages**.
MapUserPhysicalPages	Maps previously allocated physical memory pages at the specified address within an AWE region.
MapUserPhysicalPagesScatter	Maps previously allocated physical memory pages at the specified address within an AWE region.

E. Heap Functions

The following are the heap functions.

Function	**Description**
GetProcessHeap	Obtains a handle to the heap of the calling process.
GetProcessHeaps	Obtains handles to all of the heaps that are valid for the calling process.
HeapAlloc	Allocates a block of memory from a heap.
HeapCompact	Coalesces adjacent free blocks of memory on a heap.

HeapCreate	Creates a heap object.
HeapDestroy	Destroys the specified heap object.
HeapFree	Frees a memory block allocated from a heap.
HeapLock	Attempts to acquire the lock associated with a specified heap.
HeapQueryInformation	Retrieves information about the specified heap.
HeapReAlloc	Reallocates a block of memory from a heap.
HeapSetInformation	Sets heap information for the specified heap.
HeapSize	Retrieves the size of a memory block allocated from a heap.
HeapUnlock	Releases ownership of the lock associated with a specified heap.
HeapValidate	Attempts to validate a specified heap.
HeapWalk	Enumerates the memory blocks in a specified heap.

F. Virtual Memory Functions

The following are the virtual memory functions.

Function	**Description**
PrefetchVirtualMemory	Prefetches virtual address ranges into physical memory.
VirtualAlloc	Reserves or commits a region of pages in the virtual address space of the calling process.
VirtualAllocEx	Reserves or commits a region of pages in the virtual address space of the specified process.
VirtualAllocExNuma	Reserves or commits a region of memory within the virtual address space of the specified process, and specifies the NUMA node for the physical memory.
VirtualFree	Releases or decommits a region of pages within the virtual address space of the calling process.
VirtualFreeEx	Releases or decommits a region of memory within the virtual address space of a specified process.

VirtualLock	Locks the specified region of the process's virtual address space into physical memory.
VirtualProtect	Changes the access protection on a region of committed pages in the virtual address space of the calling process.
VirtualProtectEx	Changes the access protection on a region of committed pages in the virtual address space of the calling process.
VirtualQuery	Provides information about a range of pages in the virtual address space of the calling process.
VirtualQueryEx	Provides information about a range of pages in the virtual address space of the calling process.
VirtualUnlock	Unlocks a specified range of pages in the virtual address space of a process.

G. Global and Local Functions

The following are the global and local functions. These functions are provided for compatibility with 16-bit Windows and are used with Dynamic Data Exchange (DDE), the clipboard functions, and OLE data objects. Unless documentation specifically states that a global or local function should be used, new applications should use the corresponding heap function with the handle returned by **GetProcessHeap**. For equivalent functionality to the global or local function, set the heap function's *dwFlags* parameter to 0.

Function	**Description**	**Corresponding heap function**
GlobalAlloc,LocalAlloc	Allocates the specified number of bytes from the heap.	**HeapAlloc**
GlobalDiscard,LocalDiscard	Discards the specified global memory block.	Not applicable.
GlobalFlags,LocalFlags	Returns information about the specified global memory object.	Not applicable. Use**HeapValidate** to validate the heap.
GlobalFree,LocalFree	Frees the specified global memory object.	**HeapFree**
GlobalHandle,LocalHandle	Retrieves the handle associated with the specified pointer to a global memory block. This function should be used only with OLE and clipboard functions	Not applicable.

	that require it.	
GlobalLock,LocalLock	Locks a global memory object and returns a pointer to the first byte of the object's memory block.	Not applicable.
GlobalReAlloc,LocalReAlloc	Changes the size or attributes of a specified global memory object.	**HeapReAlloc**
GlobalSize,LocalSize	Retrieves the current size of the specified global memory object.	**HeapSize**
GlobalUnlock,LocalUnlock	Decrements the lock count associated with a memory object. This function should be used only with OLE and clipboard functions that require it.	Not applicable.

H. Bad Memory Functions

The following are the bad memory functions.

Function	**Description**
BadMemoryCallbackRoutine	An application-defined function registered with the**RegisterBadMemoryNotification** function that is called when one or more bad memory pages are detected.
GetMemoryErrorHandlingCapabilities	Gets the memory error handling capabilities of the system.
RegisterBadMemoryNotification	Registers a bad memory notification that is called when one or more bad memory pages are detected.
UnregisterBadMemoryNotification	Closes the specified bad memory notification handle.

VIII. CONCLUSION

This paper details the importance of virtual memory for Windows memory management. The paper highlights a major advantage of virtual memory, which is that it allows more processes to execute concurrently than might otherwise fit in physical memory. The paper aptly defines the various functions for using, allocating and even freeing Virtual Memory, comparing memory allocation methods and details some of the key Memory Management Functions.

REFERENCES

[1] http://msdn.microsoft.com/en-us/library/windows/desktop/aa366525(v=vs.85).aspx

[2] http://www.intellectualheaven.com/Articles/WinMM.pdf

[3] http://blogs.technet.com/b/askperf/archive/2007/02/23/memory-management-101.aspx

****