Writing Topic 3: Memory Management

Operating Systems II

Spring 2017

Joseph Struth

June 7, 2017

I. Abstract

Storage and retrieval of data is an important aspect of interacting with any modern operating system. This writeup will cover the differences and comparisons between how modern operating systems like Windows, FreeBSD, and Linux handle and implement memory management.

II. Windows

For memory management, similar to the Task Scheduler and I/O manager discussed in the previous writing assignments, Windows uses the Windows memory manager. The windows memory manager maps virtual address space to physical memory, and it pages memory to a disk when virtual memory use exceeds physical system memory [1]. Similar to other windows kernel utilities the memory manager is not in one single location but rather made up of multiple elements [1]. The memory manager is made up of executive services for allocation of memory, management of virtual memory, hardware memory exceptions, process and stack swapping, modifier and mapping page writer, segment defense, and the zero page [2]. Now the first function of the memory manager for virtual address space divides virtual address space into: free, reserved, shareable, or committed (private) memory [2]. Private memory can't be shared between processes. To allocate virtual memory in windows the `VirtualAlloc` function or a variation like `VirtualAllocEx` is used to reserve virtual address space and make it private [3]. This will also enable threads to reserve a set of contiguous virtual address space. If memory needs to be shared between processes and the kernel it will used shared memory. This type of memory will be accessible by one or more processes and exists in one or more virtual address spaces. This shared address space is encapsulated in the section object data structure. Each section object will be associated with committed memory or an open file that is on a disk. Windows uses the `CreateFileMapping` function to construct a section object [3]. Mapped shared address space is very useful for I/O operations to files. The second function of the windows memory manager will try to protect processes from corrupting or accessing the address space of a different process or of the operating system itself [1]. It will do this by separating user and kernel memory spaces so that user area threads can't access kernel address space. Also it will allocate a private memory space for each process, which is protected from access by other threads in different processes. Next it has some form of memory protection at the hardware level provided by the processor. The final protection is that designated shared memory has an access control list that will control when a process tries to access a different section object.

III. FreeBSD

FreeBSD is similar to Unix in many ways, including how it handles memory management. FreeBSD allocates a private address space for each individual process which is divided into three different parts [4]. First is the text segment which is used for machine readable instructions of the program (these are read only). This is where the actual code is stored. Next is the data segment which is used for both initialized and uninitialized data for the program. Lastly the stack segment is used for the run-time stack of the program. Both of the data and stack segment are both read and write-able memory [4]. To handle shared memory FreeBSD uses the `mmap` function [5]. This will let processes request to share a mapping of a file into their address space. So that one or more processes can access a file when it is modified. Since the file is shared the modifications are shown in each process that the shared file has been mapped to [5]. A similarity that both Linux and FreeBSD share is the use of functions similar to `malloc` and `free` called `zalloc` and `zfree` which allow memory allocation and deallocation [6]. In FreeBSD memory allocation takes the desired size of the memory needed as a parameter and will allocate it without using paging. The deallocation or free method uses a pointer to the memory being freed as a parameter.

IV. Comparison to Linux

Linux is different than both FreeBSD and Windows in that it represents each physical page on the system using a structure called a page structure [7]. The page structure holds information like the page status, usage count, virtual addresses of the page, and meta-data information [8]. The Linux kernel uses this page structure to track all of the pages in the operating system which helps manage when pages are freed and which process have what pages. To further manage pages Linux uses page zones to differentiate pages. The first zone is used for direct memory access or DMA this is the `ZONE_DMA`. Next is the direct memory access zone specifically for 32 bit devices called `ZONE_DMA32`. Normal pages are in the `ZONE_NORMAL`. Lastly the pages that are not permanently in the kernels address space are in the `ZONE_HIGHMEM` [9]. Linux, FreeBSD, and Windows all are similar in that they use some form of paging to go from disk to virtual memory. The operating systems are different in how they implement the data structures and sharing between processes.

References

[1] M. Russinovich, D. Solomon and A. Ionescu, Windows Internals, Part 1, 1st ed. Boston, USA: Microsoft Press, 2012.

[2] About Memory Management, MSDN. [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/aa366525(v=vs.85).aspx. [Accessed: 07-Jun-2017].

[3] Using the Memory Management Functions, MSDN. [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/aa366885(v=vs.85).aspx. [Accessed: 07-Jun-2017].

[4] M. K. McKusick, K. Bostic, M. Karels, and J. Quarterman, The Design and Implementation of the 4.4BSD Operating System, FreeBSD.org. [Online]. Available: https://www.freebsd.org/doc/en/books/design-44bsd/. [Accessed: 07-Jun-2017].

[5] 2.5. Memory Management Chapter 2. Design Overview of 4.4BSD, FreeBSD.org. [Online]. Available: https://www.freebsd.org/doc/en/books/design-44bsd/overview-memory-management.html.

[6] M. Dillion, Chapter 7. Virtual Memory System Part I. Kernel, FreeBSD.org. [Online]. Available: https://www.freebsd.org/doc/en/books/arch-handbook/vm.html.

[7] D. Rusling, Chapter 3 Memory Management, The Linux Kernel. [Online]. Available: http://www.tldp.org/LDP/tlk/mm/memory.html. [Accessed: 07-Jun-2017].

[8] J. Corbet, A. Rubini and G. Kroah-Hartman, Linux device drivers 15.1. Memory Management in Linux, 1st ed. O'Reilly, 2010.

[9] A. Ott, Virtual Memory and Linux Linuxfoundation.org. [Online]. Available: http://events.linuxfoundation.org/sites/events/files/slides/elc_2016_mem_0.pdf.