Writing Assignment Three

Brennan Giles

CS 444 Oregon State University

Fall 2018

Abstract

Memory keeps the workflow going by allowing users to save and load data, and thus is an essential component of any computers functionality. The trouble is keeping all the memory sorted properly so processes dont get mixed up or saved data corrupted. In this paper, we will investigate the different ways Windows, FreeBSD, and Linux handle memory so that college students can save those papers they worked on for hours.

1 WINDOWS MEMORY

Each process on 32-bit windows has its own virtual address space that allows addressing memory, up to 4 gigabytes. It is much larger for 64 bit; up to 8 terabytes! All threads of a process can access the virtual address space it was allocated, and threads cant access or steal memory space that belong to other processes in order to avoid corrupting eachother [1]. Im going to discuss the 64 bit memory architecture since it is what I use on my devices, and to start lets investigate the Kernel Address Space. The Kernal Address Space includes a system page table entry (or PTE), a Paged Pool, System Cache, and non paged pool which is used for images. It provides up to 8 TB for the kernel to work with. Lets talk about Page Tables some more; a Page Table is an internal data structure used to translate virtual addresses into their corresponding physical address. When a Page is moved in physical memory, the system updates the page maps of the affected processes. If the system ever requires more space in physical memory, it moves the least used portions of the physical memory to a Paging File. The PageFile supports system crash dumps and allows the system to use physical RAM more efficiently by writing content to a hard disk when main memory is near capacity. Without enough RAM or memory, there would be serious errors in applications so it has a very important job.[1]

Moving on, the working set of a process is the set of pages in virtual address space. The working set has only pageable memory allocations and nothing else is allowed. The pages of that working set have three potential states: Free, Reserved, and Committed. Free means the page is neither committed or reserved, Reserved means the page is reserved for future use and will likely be written to soon, and Committed means Memory charges have been allocated from the RAM and paging files on disk. To commit physical pages from a reserved region, a process can use VirtualAlloc or VirtualAllocEx. At this point you may be wondering how memory is protected as it is written and evaluated this is done through Copy on Write Protection. COW protection allows multiple processes to map their virtual address spaces such that they share a physical page until one of the processes modify the page. It allows the system to conserve physical memory and time by not performing an operation until absolutely necessary.[1]

In summary, Windows memory uses pages in order to translate between physical and virtual memory, and isn't afraid to use a little RAM in order to handle near capacity situations.

2 LINUX / FREEBSD MEMORY

Since Linux and Free BSD are both Unix based Operating systems, I will explain how both handle I/O at the same time since they are extremely similar. Both use the same idea of virtual memory in order to handle, organize, and improve the efficiency of physical memory. In Linux/FreeBSD, when a CPU decodes an instruction that accesses system memory in some way it translates the virtual address within the instructions to the actual physical address so that the memory controller knows where to find it. Just like with Windows, Linux uses pages which are sized based on architecture. Page Tables operate the same in here as they did with Windows as well, being used to describe virtual pages and what physical address they are assigned to. [2]

In order to speed up the memory address translation, Linux CPUs maintain a cache of translations called Translation Lookaside Buffer, or TLB. Linux also tries to map memory pages to higher levels of the page table that tend to hold larger sizes these are called huge pages. These reduce the pressure exerted on TLB so that the system can perform better. Linux/FreeBSD use two different methods of mapping physical memory with huge pages, which are HugeTLB filesystem and Transparent HugePages. HugeTLB filesystem uses RAM as a backing store so that it can handle all that data faster and easier. Transparent HugePages is considered newer and easier to use than HugeTLBFS because instead of requiring extensive user configuration on what parts of memory can or should be mapped by the huge pages, THP manages it transparently and automatically. [2]

Linux and FreeBSD use something called Nodes to handle memory banks with different access latencies depending on their distance from the processor. When multi-processor devices use non-uniform memory access systems they utilize nodes in order to arrange the memory into banks. Each node has its own set of zones and banks, along with various information about the data stored within. [2]

Last of all, what happens when memory is completely used up? When the Linux or FreeBSD kernel sees that the system has run out of memory, or OOM, it boots up the OOM KILLER. What this nefarious process does is kill tasks in an effort to free up memory and save system health. Once just enough is freed to allow smooth operation again, defragmentation and reclamation is usually invoked to clean up space and ensure some elbow room is afforded any processes that come next. [2]

3 Comparison

Unlike the last two papers Linux, FreeBSD, and Windows have extremely similar methods of handling memory. Luckily there are a few key difference we can discuss that stand between them, for example the data structures used for each. While Windows uses tree data structure in its memory systems, Linux/FreeBSD use Linked List. Linux maintains a list of vm-area-structs which is searched whenever a page needs to be found. Once entries grow greater than 32, the linked list is converted into a tree, and this results in a very adaptable and efficient system. Windows uses VAD, or Virtual Address Descriptors for each node of its tree and these VAD nodes are marked as free, committed, or reserved like discussed above. [3] The way Windows handles Data structures is not as dynamic as Linux but sometimes Simplicity proves very beneficial. Another key difference between the two is that while Windows uses cluster demand paging (eight pages brought in at once), Linux uses demand paging with no pre paging applied. Linux uses a lazy swapper and never swaps a page into memory unless needed. Last of all (and we could go on for ages about the minute differences), Windows uses FIFO for Page replacement while Linux uses LRU or Least Recently Used. Here, Linux is clearly superior in that the weakest pages are consistently replaced vs. Windows where the queue is used, which results in poor performance and page fault rate increases. [3]

In the end, Both Operating systems share a lot of similarities and do their best to provide the best user experience when it comes to memory. Linux/FreeBSD have an unmistakable advantage however when it comes to memory management and cleanup, most likely the result of decades of community and professional involvement in what must be the most ironically unmemorable components of an operating system.

References

[1] Windows Hardware Dev Center. About Memory Management May 30, 2018.
`https://docs.microsoft.com/en-us/windows/desktop/memory/about-memory-management`

[2] Kernel Development Community The Linux kernel users and administrators guide August 13, 2010.
https://www.kernel.org/doc/html/latest/admin-guide/README.html

[3] UK Essay Comparison of the memory management of Windows with Linux December 5, 2016.
`https://www.slashroot.in/linux-system-io-monitoring`