I want you to look at memory management. Examine how Windows and FreeBSD implements it, and compare each to Linux. Specifically, how do they differ? How are they the same? Why do you think these similarities or differences exist?

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 don’t 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.

Windows

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 can’t access or steal memory space that belong to other processes in order to avoid corrupting eachother [1]. I’m going to discuss the 64 bit memory architecture since it is what I use on my devices, and to start let’s 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. Let’s 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.

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.

It wouldn’t be a computer science paper if we didn’t mention the heap, so let us investigate how windows approaches it. To start, each process has a default heap provided by default. Applications that make frequent allocations from the heap are recognized and eventually are assigned private heaps. Private Heaps don’t allow anything other than the process that created it to attain access, but operate in the same way as normal heaps.

Linux/ Free BSD

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.

Comparison

FreeBSD and Linux are almost exactly the same in how they handle I/O. Both use Blocks that allow access randomly or sequentially, and both utilize an “I/O stream” to receive input and interpret it. Both also use the three descriptors of standard input, output, and error to handle processes as well. Both systems are extremely lightweight and efficient, which results in tailor made and exceptional user experiences when it comes to interaction. Its one major weakness is that while it does allow for asynchronous processing thorugh threading, it struggles with large amounts of concurrent I/O. Each thread requires a stack, consuming large amounts of memory and the threads that can be used on any process are often limited.

Windows on the other hand doesn’t rely on the cyclic nature of Unix data structures, and instead has a central overlord, the “I/O manager” that handles the majority of Input and output operations. The I/O manager provides an efficient and effective method of handling I/O streams, while simultaneously providing better safety nets for data being transferred. Furthermore, Windows uses a priority system that enables the effective use of I/O to clear up room on the RAM in cases of low memory or other special circumstances. This results in a more stable environment, especially under memory intensive situations.