%Intro

Memory management within FreeBSD, Linux and Windows is the process of creating an address that directly references a specific area of physical memory. This process involves Random Access Memory, Flash Based memory, and memory caches\cite{Whatisme93:online}.. These abstractions of the physical memory are often just kept within the virtual memory space in the form of addresses. For the Linux platform, this is usually done with pages and paging tables while Windows decides to use a differently structured approach, however, they leave the overall semantics of the concepts relatively similar across the board. In FreeBSD, Linux, and Windows memory management will involve the allocation and reallocation of data to specific blocks of memory while it organizes it into a comprehensible form for a program to quickly access and scoop up needed info, this can also be useful readability for programmers who work on kernel level applications.

%FreeBSD to Linux

As with the Schedulers and the General Input/output, FreeBSD and Linux are very similar, as many of their core components from the same ancestor. There are exceptions in how they behave however, that set FreeBSD and Linux apart from each other. In FreeBSD, if memory is limited for a given system then it will completely swap a process into its swap memory on the hard drive. This context swapping method isn’t unique to FreeBSD alone, FreeBSD and Linux also utilize very similar naming conventions for their respective malloc and freeing functions, and example being ‘zalloc’. Zalloc is traditionally used in embedded devices, however, they serve as an east approach for mallocing and freeing space with zeros instead of doing it manually within the kernel or even within the user level\cite{freeBsdBook}.

Where the unique aspects start to show in regards to memory management is in the case when memory is moderately limited for all processes on the system, the kernel for FreeBSD will then tune shared amounts of memory to each process within the system attempting to evenly distribute it across all physical memory\cite{25MemoryMan:online}. Exceptions can be made to this though, if on initialization or at any point during its runtime a process tells the kernel how much memory it wants in the future, FreeBSD will do its best to accommodate, aiding in the overall performance of the system. Memory is a relatively expensive tool, is not often used for the FreeBSD system, whereas connecting discs are large and fast. Thus, frugality in memory usage had been applied, which favored sacrificing extra disc input and output to be used for the system, instead of using memory. To avoid completely running out of memory for a given process, FreeBSD allows processes and threads the ability to share parts of mapped virtual memory space within the Random Access Memory. This results in any change by a single process within the kernel to be visible to another process that shares the data, this also applies vice versa, this system had been implemented in order to slowly replace the older system of limiting memory to all processes overall.

%Windows to Linux

Inside of the Windows kernel, memory management is approached slightly differently from the linux kernel. The first major difference between the two platforms being page sizing. Within Windows, pages can be one of two sizes: large (2MB), or small (4KB)\cite{windowsbookpt1}. Linux on the other hand (varying on architecture and implementation of the system of course) decided to use 4KB and 8KB page sizes\cite{Linux\_Book}. The pages for either system are managed in a similar way concept-wise, through the use of paging directories and tables. Address mappings for memory are structured somewhat like a tree, branching from the root to different tables depending on the index of the address space itself. This structure intends to minimize search times for a page, and organizes addresses in a uniform way for kernel level developers who are creating drivers or kernel level tools.

While Pages within Linux are effectively the spine of data management that can be manipulated by drivers, Windows opts to require several other services including page writers, managers for working sets, process and stack swappers, as well as a thread dedicated to writing zeros to a page (because other methods had given significant impacts to the performance of the Windows Kernel). Pages in Windows also come in a variety of states, whereas Linux for the most part uses a first come serve basis depending on the process that is allocating memory. The states for the Windows kernel are as such: Free (meaning allocated, but unused), Reserved (Meaning that it will be allocated and used, this is a preemptive approach to the memory allocation for a process.), Committed (this is private towards a specific process, and the page is allocated), and shareable (this is non-private mapped and can be accessed by any other process or thread). These states all asist pages in windows to help setup the priorities for allocation of processes.

%Conclusion

The same general concepts seem to be applied for each of our platforms, FreeBSD, Linux, and Windows all appear to implement roughly the same concepts behind memory management. All platforms implement a form of memory address indexing within virtual memory space that serves as a middle man between page allocations and physical memory. This management system abstracts to lower levels of architecture put forth by the kernel, as well as the digital logic behind keeping the process data. All three platforms implement forms of paging that will hold all the quickly accessible data for kernel level processes. All three platforms also implement the same functionality of allocation, and reallocation of pages. Differences start to arise within FreeBSD in its approach to handling limited physical memory by limiting resources to all processes, and sharing memory addresses between the processes. The differences within Windows take a different approach to the sizes of the pages themselves (This goes down to naming conventions of huge, vs large, vs small). However, all three platforms share some nearly exact similarities in their ability to use the mapping library in order to index files and the contents of data within varying address spaces within the kernel to assist in faster access.