Writing Topic 3: memory management

Behnam Saeedi
CS444: Operating systems II

Spring 2017

Abstract

This document will be covering the memory management features of the three operating systems discussed in this class. These three operating systems are Linux, Windows and FreeBSD. This document will cover some of the details about how the memory management features implemented, what are some of their features and how do they stack up against Linux's implementation of those features.

Contents

1 Linux

1.1 Memory Management

Due to size limitations on the amount of memory available to a kernel it is extremely important to have a subsystem dedicated to organize and manage distribution and handling of memory. This task falls under Kernel's responsibilities under Kernel utilities known as Memory management tools. This section will be covering the basics of Linux Kernel's memory management tools in depth. Furthermore, it will proved a brief description of Pages, Services, Features and capabilities of Linux Kernel's memory management.

1.2 Services

1.2.1 VM

Linux provides the memory to processes at the same time through a concept known as virtual memory. This means every process has access to all of the memory at the same time giving it the illusion of a much larger memory than the actual physical RAM.[1]

1.2.2 Address Space Protection

Linux Kernel Guarantees that each memory gets it own virtual address space. Each virtual address space is completely independent of others and is achieved through offsets in memory.[1]

1.2.3 Memory Mapping

In Linux, files could be mapped directly to to a processes virtual address space. This process is known as memory mapping. This feature is one of the requirements for the memory management and is present in Linux, Windows and FreeBSD.[1]

1.3 Pages

Since the size of the virtual memory is significantly larger than the physical memory there is a need for a reliable way to address the memory. The process of assigning the virtual memory to the physical memory is known as paging. Pages are 4KB units of contiguous memory on the RAM and the basic unit of memory known by both Kernel and the CPU. This is regardless of the fact that both Kernel and CPU do have access to individual bits written on each page.[2]. The operating system could fall into two dangerous scenarios: Data Collision and unused data. These two problems are the two extremes of the problems that can occur. memory collision happens when one physical memory is addressed by more than one virtual memory. The second issue occurs when a physical memory is not addressed by any virtual memory of running processes and is not being used efficiently.[1]

1.3.1 Efficiency and demand paging

One way of preserving efficiency of the program is to avoid allocation of virtual memory while the process is not running. This is known as Demand Paging. Linux uses this method to take care of file load and maps it to the memory.[1]

1.4 Features

Linux kernel provides a set of features that allow users to fully take advantage of the provided memory.

1.4.1 Cache

As discussed in the class, in order to have the programs run more efficiently, concept of caching is introduced. Task of this feature is to keep a copy of the data for future use with a faster access time than the storage drive. This process is known as caching and it can take place on several different levels.

- Buffer Cache: Block device driver's data buffers
- Page Cache: Memory map speed boost (allowing faster file access)
- Swap Cache: file data that is going to be kicked off of the memory. (dirty pages only)
- Hardware Cache: Translation, look-aside buffers [1]. CPU will not check the entire memory every time it needs access to data.

1.4.2 Swapping

What if a process requires a memory and there are no more available memory to be allocated for the process? In this case the Kernel has the important task of opening up more memory. In order to deal with this the kernel will select a process that is unlikely to be used soon again and places it on a different storage (drives). Then the recently emptied memory could be used for memory allocation.[1]. There are several algorithms available for kernel to make the important decision of which process needs to be swapped. The current algorithm is LRU or Least Recently Used. Kernel looks for each process and if that process has not been used for a long time, it gets swapped out from the memory. Lets assume that kernel makes a bad decision in swapping a process and every time it swaps a process, that process gets requested. This will lead to an issue known as thrashing. In this scenario, the performance suffers significantly since the entire CPU's execution time gets limited to the speed of the storage device.

1.4.3 Threads

IEEE Computer Society has series of standards that are specific to compatibility across multiple operating systems. These standards are known as The Portable Operating System Interface or POSIX. One set of libraries provided by POSIX are known as the thread libraries for c and C++. These libraries operate based on the concept of concurrent processing. In Linux, this concept allows multiple programs to be run in a way that gives the illusion of multiple programs running at the same time to the user. It is important to consider that each CPU package still handles each process one at the time per physical core. In Linux threads can be create by a parent process. This concept is further described in the Processes portion of the writing assignments.[3].

1.4.4 Shared Virtual memory

As covered earlier, process has the capability of creating threads. This however, poses an issue. The parent process needs to be able to monitor the data processed by each thread. In order to accommodate with this need, threads in Linux share the same address space with other threads of the same parent and the parent itself.[1]. Linux requires each process and thread that shares memory to have its physical page frame to be stored in a page table entry stored at all sharing processes and threads.[1]

1.4.5 Access Control

Based on what was covered on Shared Virtual Memory (1.4.4), The page table needs to contain the access control information for each page. As a side effect, now the process can also see who else has access to a specific memory space. This information could be found in the following lookup table (refbitfields) this table was found at source [1].

Table 1
Meaning of bit fields and OS control on access.[1]

Field	Meaning
V	Valid, if set this PTE is valid,
FOE	"Faulton Execute", Whenever an attempt to execute instructions in this page occurs, the processor reports a page fault and passes control to the operating system,
FOW	"Faulton Write", as above but page fault on an attempt to write to this page,
FOR	"Faulton Read", as above but page fault on an attempt to read from this page,ASM
ASM	Address Space Match. This is used when the operating system wishes to clear only some of the entries from the Translation Buffer,
KRE	Code running in kernel mode can read this page,
URE	Code running in user mode can read this page,
GH	Granularityhint used when mapping an entire block with a single Translation Buffer entry rather than many,
KWE	Code running in kernel mode can write to this page,
UWE	Code running in user mode can write to this page,

1.4.6 Paging tables

In Linux the paging happens through three page tables. These tables are Level 1, Level2 and Level 3. This is achieved by having each level providing an index number containing the frame for the following level similarly to what is described in Intel architecture but with only 3 levels.

2 WINDOWS

2.1 Memory Management

Windows provides its own series of services and features for memory management. In this section we will cover some of these services and features.

2.2 Size

In windows the 32-bit processes take advantage of 2 GB of memory by default. This size could be increased to 3/4 GB on 32/64-bit Windows distributions. Furthermore 64-bit processes can take advantage of up to 8182 GB of virtual memory space.

2.3 Services

The Windows kernel provides the following services to the user for memory management:

- Address Mapping
- Paging
- Memory Mapped Files
- Copy on Write Memory
- Physical Memory Allocation and Use.

Furthermore,On working set (Set of pages physically present), there are 5 key parts:

1) Working Set Manager
2) Process / Stack Swapper
3) Modified Page writer
4) Mapped Page Writer
5) Zero Page thread: This thread is always running and has the task of setting the memory to zero. This happens on a hardware level and is very fast.

2.4 Pages

The Windows memory locking feature is much more fine grain than both Linux and FreeBSD even on memory subsystem level. This is in comparison to Linux and FreeBSD where they have a Kernel level lock. Windows provides 2 page sizes: Large (2 MB) and Small (4 KB). Furthermore, These pages can classify under 4 states:

- Free: The memory is free to be used.
- Reserved: Memory is requested but it is not yet being used. In other words, that memory is private.
- Committed: This memory falls under Private and valid mapping.
- Sharable: This memory is similar to Committed with the only difference being that this memory is not private.

2.5 Features

In Windows, Process acts as a container which holds threads. This is explained further in the Processes section of writing assignments. To recap. These processes fall under a process group and each at least have to have one thread.These processes could create heap, stack, or a combination of any number of them if the developer choses to. This provides a unique opportunity for the process to switch its stack on the fly as the program is running. Finally windows has a unique feature called Fibers. Fibers are user level scheduled abstract processes that the developer needs to take care of its scheduling. From the kernel's perspective all of fibers within a process are a single thread.

2.6 Comparing to Linux

There are several major differences between how Linux and Windows handle the memory. Windows has a completely different construct for the processes. These differences are in Windows' advantage since they have managed to come up with a fine tune setup in order to improve the kernel's performance. Linux has a rather more simplistic but effective approach towards these problems. Windows provides more options and a more developer oriented approach towards these issues.

3 FreeBSD

3.1 Memory Management

The FreeBSD memory management is very similar to Linux. In fact many of the provided features and services are based on Linux libraries and some are improved on. In general Linux and FreeBSD's differences are mostly in licensing. FreeBSD's developer team is much slower.

3.2 Pages

In FreeBSD, similarly to Linux and Windows, memory is managed through a page by page basis. [4]. This means similarly to Windows and Linux the smallest unit of memory that could be accessed is 1 page or 4 KB of contiguous chunk of memory. The memory itself is accessible down to a single bit, but the handling of memory happens at a page level.

3.2.1 States of memory

According to FreeBSD manual, in FreeBSD, memory could be in one of the following states at any given time:

- Wired
- Active
- Inactive
- Cache
- Free

The memory in all states except Wired, are stored in a doubly linked list. Wired pages are not stored on any queue. /cite4. FreeBSD provides more involved paging queue for pages that classify as Cache and Free states with comparison to Linux.

3.2.2 Swapping

FreeBSD swaps out entire idling processes. This is unlike Linux where only least recently used gets swapped out. It takes advantage of a construct known as Global-LRU across all of user pages. Another used construct for FreeBSD for swapping is page-coloring optimization. This optimization achieves a more accurate method of swap selection by incorporating the processor's cached pages. (See Hardware Cache at 1.4.1).[5].

3.3 Services and features

FreeBSD provides a very similar constructs and services to Linux. In this subsection we can look into few differences that FreeBSD and Linux have.

3.3.1 Generic VM Object

FreeBSD provide and idea known as generic virtual memory objects. Virtual memory objects could be unbacked, backed with swap, physical device or file storage. This results in Unification of the buffer cache (1.4.1) in FreeBSD. [4]. FreeBSD dynamically tunes page queues in order to balance the pages in for maintaining a reasonable breakdown of clean and dirty pages. [4].

3.4 FreeBSD and Linux Comparison

As mentioned earlier, FreeBSD and Linux have many similarities on abstract level features. However, FreeBSD usually improves on the Linux implementations. These improvements come at a cost of FreeBSD having a much slower development.

References

[1] "Memory management," http://www.tldp.org/LDP/tlk/mm/memory.html, accessed: 2017-06-06.
[2] "Paging and swapping," http://www.linux-tutorial.info/modules.php?name=MContent&obj=page&pageid=89, accessed: 2017-06-06.
[3] "Posix thread (pthread) libraries," http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html, accessed: 2017-06-06.
[4] "Management of physical memory," https://www.freebsd.org/doc/en/books/arch-handbook/vm.html, accessed: 2017-06-07.
[5] "A comparison of the memory management sub-systems in freebsd and linux," file:///C:/Users/Behnam/Downloads/Documents/aa5e870fdcb525abf7ed5ca40f58cbb007a3.pdf, accessed: 2017-06-07.