CS 232
Operating Systems

Assignment 2: Understanding Memory Virtualization Holistically

MUHAMMAD MUNAWWAR ANWAR (MA04289)

November 01, 2020

Habib University

Contents

List of Figures

Introduction

The aim of this report is give a brief summary of chapter 23 of our book titled "Complete Virtual Memory Systems " which details how a complete virtual memory system is put together. This chapters covers two systems, VAX/VMS and Linux on Intel x86.

VAX/VMS

VAX/VMS Address Space

The VMS provided a 32-bit virtual address with 512-byte pages per process. The virtual address was then further divided into a 23-bit VPN and a 9-bit offset. The VMS was a hybrid system that used both segments and page tables, so the upper two bits of the VPN were used as segment bits. See [2, fig.1]. The address space was divided into four segments, P0, P1, System(S) and unused.See [2, fig.2]. P0 and P1 refer to the process space which is unique and used for each process. The base register holds the address of page table for segment and the bounds register holds the size of the page table. The protected OS code and data lies in the segment S which is shared across processes.

Page Replacement

The page table entry in VAX/VMS contains the following bits : a valid bit, protection bit (4-bits), dirty bit, a field reserved for the OS use (5-bits) and physical frame number (PFN).

The VMS uses a segmented FIFO replacement policy along with two second-chance lists where pages are placed before being evicted from the memory, more specifically a global clean-page free list and dirty-page list. Each process has resident set size (RSS), the maximum number of pages it can keep in the memory. Pages are kept on a FIFO and when a process exceeds its RSS, the "first-in" page is evicted and placed on the end of the clean-page list if clean and on the end of the dirty-page list if dirty. If a process needs a free page, it takes the first free page off the global clean list. If a process faults on a page in the second chance list, it reclaims that page from the either the clean or dirty list .

VMS Lazy Optimizations

Clustering

The VMS groups large batches of pages together from the global dirty list and writes them out to disk together. After the pages are written out to the disk, they can be marked clean.

Demand-zeroing

When a page is added to the page table, it is marked as inaccessible. If the process accesses the page, a trap into the OS takes place. OS then finds a physical page, zeros it, and maps it to the processes virtual address space by updating the PTE. If the process never accesses that page, no work is done. See [1, fig.3].

Copy-on-write(COW)

Instead of copying the page, mark the physical page as read only. Have the virtual page in both address spaces map to the read-only physical page. If both pages never write to the page, no further work is done. If the one of the address spaces does try to write to the page, trap into the OS , which sees the page marked copy-on-write. Then the OS can allocate a new page, fill with data and update the PTE. See [1, fig.4].

LINUX

Linux Address Space

The address space is divided into the user and kernel space. Each process has its own user space whereas the kernel region is same across processes. The kernel space is divided into logical and virtual regions. The Logical address space is mapped directly to the physical i.e. contiguous regions in logical memory are contiguous in physical memory. The virtual address can be mapped any to physical page, i.e. contiguous regions in the virtual memory are not necessarily contiguous in the physical memory. See[3,fig.5]

Page Table Structure

Intel x86 provides a hardware managed, multi-level page table structure with one table per process. The OS points a privileged register at the start of the page directory, and the hardware handles the rest. The OS gets involved at process creation, deletion and upon context switches to ensure that the correct page table is being used by the hardware MMU to perform translations. The Intel x86-64 uses a four-level table . Only the bottom 48 bits out used. Out of the 48 bottoms, 12 bottom bits are used, leaving 36 bits for Address translation. See [4,fig.6]

Large Page Support

Intel x86 allows for multiple page sizes in the Hardware, reducing the number of mappings in the page table. Moreover, it reduces the pressure on TLB. Applications explicitly require memory with large pages via mmap(). More recent versions of Linux have added transparent large page support, automatically looking for opportunities to allocate 2MB pages without application modification.

Page Cache

The Page Cache keeps pages from sources : Memory-mapped files, File data and Heap and Stack pages. Page Cache keeps track if the entries are clean or dirty. Dirty pages are periodically written back to the disk by background threads. After a certain period or too many pages are dirty, Linux evicts pages using 2Q replacement algorithm.

In 2Q replacement algorithm, keep two lists and divide the memory between them. When a page is accessed for the first time it is put on the active list. When it is re-referenced , the page is promoted to the active list. When pages are needed, pages are evicted from the inactive list. The active list is about 2/3 total cache page size.

Security

Buffer Overflow

The defense against buffer overflow is to prevent execution of any code found within the stack. NX bit (AMD) and XD bit (Intel) prevents execution from any page that has bit in the PTE.

Return-oriented Programming

Return-oriented programming(ROP) strings together pieces of existing code called gadgets. The Linux defends against ROP by performing Address space layout randomization (ASLR). The OS randomizes the placement of code, heap and stack each time a program is run. This makes it almost impossible to predict the addresses of the gadgets in memory.

Meltdown and Spectre

These attacks reveal memory execution by taking advantage of speculative execution down by the processor. As a Linux uses Kernel Page Isolation table (KPTI). In KPTI, remove the much of the kernel space from each user space and have separate kernel page table for most kernel data. Kernel code and data structures are no longer mapped into each process. When switching into the kernel, a switch to the kernel page table is needed.

Conclusion

In the above discussion, we saw how key elements of memory virtualizations come together to form a complete virtual memory system. We also saw how the goals of the OS designers and the challenges they faced changed over time . But then we also saw innovations that were made to maintain the abstractions of the OS.

Innovation

Reclaiming Pages

The VMS uses a local replacement policy i.e. Segmented FIFO policy (SFIFO). The SFIFO improves the performance of FIFO by providing second-chance lists and thereby reducing costly disk accesses. On the other hand, if the clean-page free list were to be empty whenever a process would request for a page, we would have to load a page from the disk which is quite a costly operation. In order to avoid this situation, we can use a different strategy.

In this strategy, we satisfy all memory requests from the clean-page free list but rather than waiting for the list to drop to zero before we begin selecting pages for replacement, we trigger page replacement when the list falls below a certain threshold. When the free memory falls below the threshold, we can start reclaiming pages from all the processes. Consequently, ensuring that there is always enough free memory to satisfy requests. Once, the maximum threshold has been reached we can suspend reclaiming pages from the free list. See [1,fig.7]

Figures

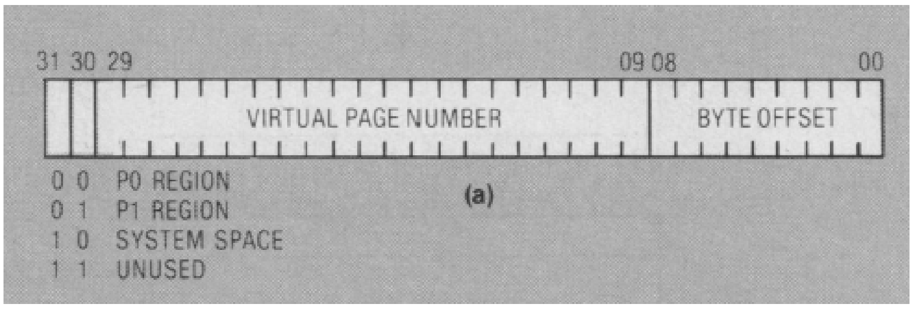

Figure 1: VAX-11 Process virtual address: Source:[2]

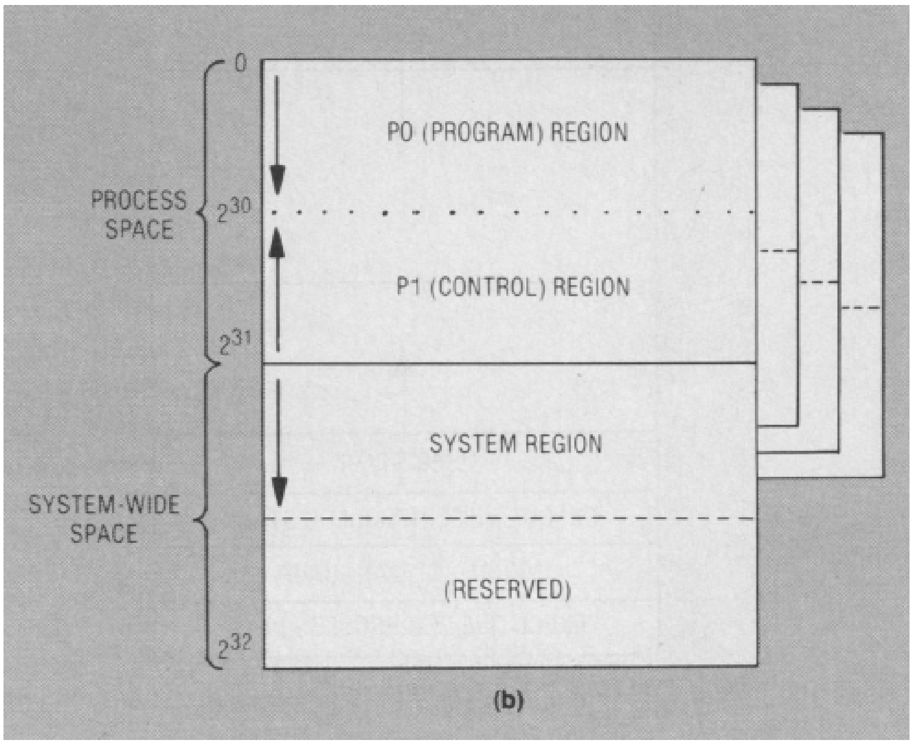

Figure 2: VAX-11 Virtual Address Space. Source:[2]

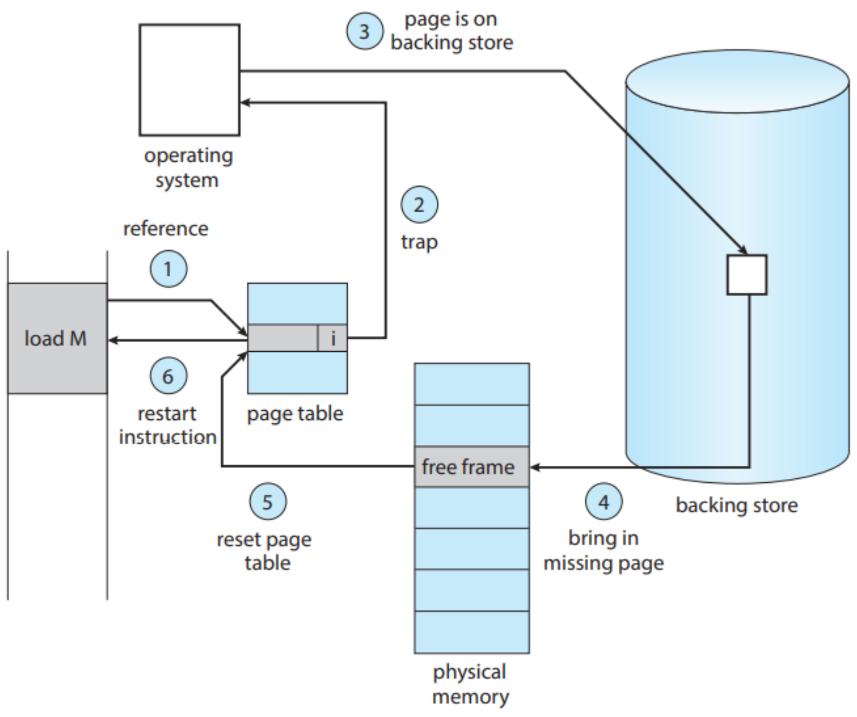

Figure 3: Demand Zeroing Pages. Source:[1]

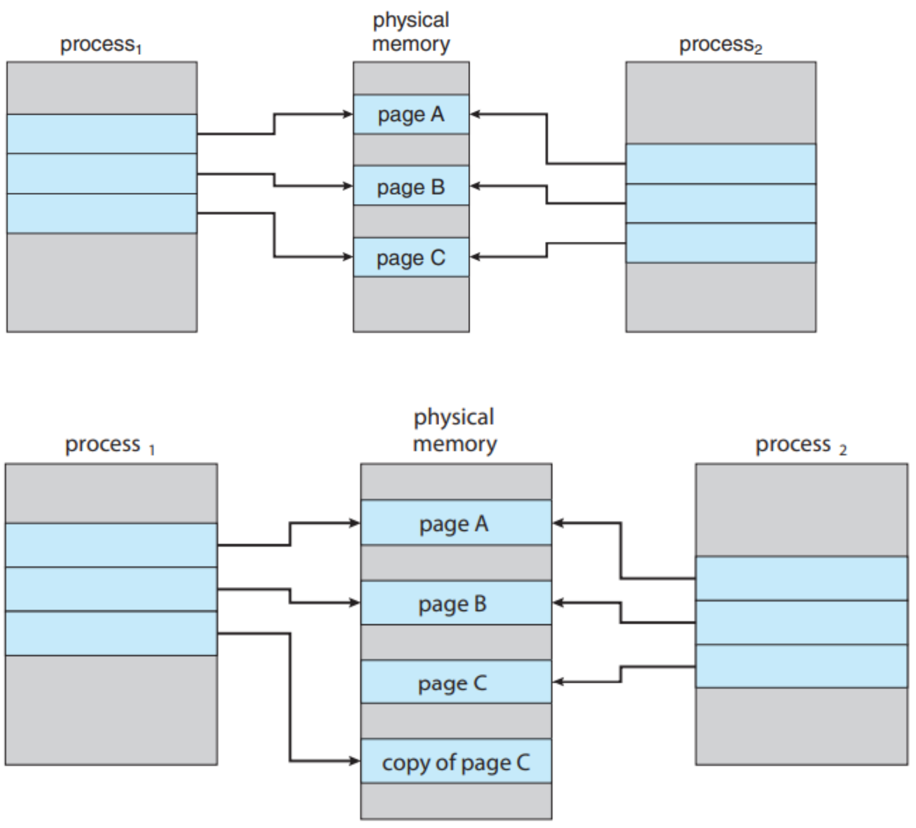

Figure 4: Physical Memory before and after Process 1 modifies Page C. Source:[1]

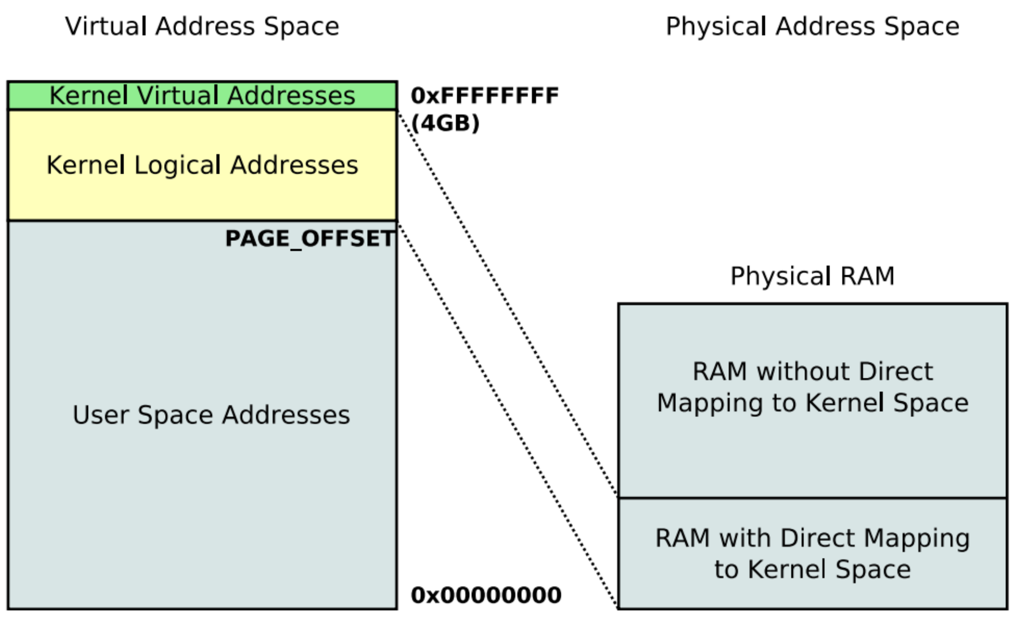

Figure 5: Linux Virtual Address Space. Source:[3]

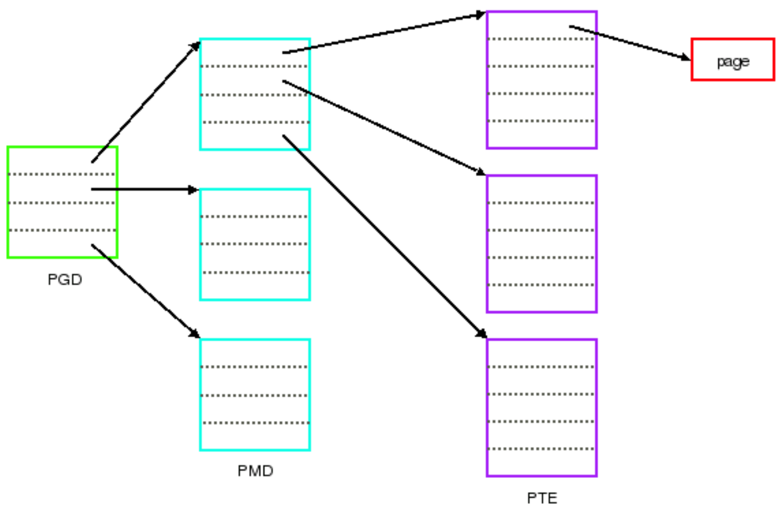

Figure 6: Intel x86-64 Multi Level Page Table. Source:[4]

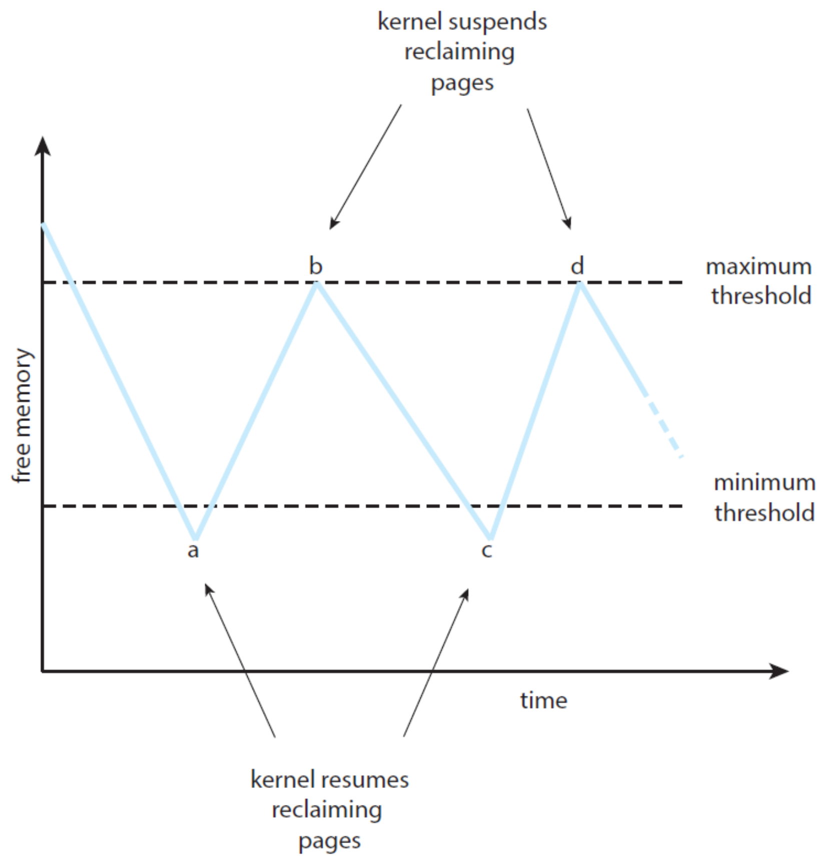
kernel suspends reclaiming pages
b
d
maximum threshold
free memory
minimum threshold
a
c
time
kernel resumes reclaiming pages

Figure 7: Reclaiming Pages. Source:[1]

References

[1] A. Silberschatz, P. B. Galvin And G. Gagne, “Operating Systems Systems Concept”, 10 Ed., Willey, 2013.

[2] H. Levy, P.Lipman, “Virtual Memory Management in the VAX/VMS Operating System”. *IEEE Computer*, vol. 15,no. 3,March.,pp.35-41,1982.

[3] A.Ott.,“Virtual Memory and Linux”,*Embedded Linux Conference*,2016[Online].Availaible: https://events.static.linuxfound.org/sites/events/files/slides/elc_2016_mem.pdf .

[4] “Four-level page tables,” [LWN.net]. [Online]. Available: https://lwn.net/Articles/106177/. [Accessed: 01-Nov-2020].

[5] Arpaci-Dusseau, R. and Arpaci-Dusseau, A., 2014. Operating Systems: Three Easy Pieces. 1st ed. [ebook] Wisconsin-Madison: Arpaci-Dusseau, pp.1-14. Available at: <http://pages.cs.wisc.edu/ remzi/OSTEP/file-disks.pdf> [Accessed 17 October 2016].