Chris Kasper
CS 370-001
Homework Set #3

Chapter 9

1. Suppose we have a hardware segmentation system like that shown in Figure 9-3 where the base and limit tables have eight entries. Base register i (numbered from 0) contains the value i × 210 + 0x100 and each limit register contains the hexadecimal value 0x200. If the 16-bit virtual address is 0x605f, then what physical address is computed? Does this address generate a fault?

Number of indexing bits in VA = 3 (since each table has 8 entries)
VA = 0x605f = 0b**110**000001011111
Limit entry = 0x200 = 0b1000000000
Base table entry 6 = 6 * 210 + 256 = 1516

PA = 1516 + 0b000001011111 = 1611 (0x064B)
0x200 (512) is not less than 95 (0b000001011111), so no fault Is generated

3. The process memory layout shown in Figure 9-7 is based on hardware that decrements the stack pointer when pushing onto the stack. How would the design change if the hardware incremented the stack pointer instead?

If you were to change the design if the hardware increments the stack pointer, instead of decrementing it, you would have to initialize the stack pointer to its lower address boundary in the memory layout, so just above the data segment. The stack almost works in reverse, with new items getting higher addresses then the items currently on the stack.

6. Consider a system with 64-bit addressing. If the page size is 64 KB, how many levels must the paging system have so that each table fits within one page? (Take the PTE size to be 8 bytes.)

VA = 64 bits
Page size = 64 KB (2^{16} bits), so offset bits = 16
Remaining bits = 48

If use three-level page table, each level is addressed by a 16-bit number (48/3), the page table would take up 64 KB.

7. If we have a machine with a 32-bit virtual address and a 1-KB page size, then how many pages are in the virtual address space? If each PTE takes 4 bytes, then how many pages are required to hold a complete page table? If the physical address space is 256 MB, then how many bits are needed for the page frame number (PFN) in the PTE?

1 KB = 2^{10} bits
# of pages in virtual address space = 2^{32-10} = 2^{22} or 4 M pages

PTE = 4 bytes (2^2 bits)
Size of page table = 2^{22} * 2^2 = 2^{24} or 16 MB
Number of pages to hold a complete page table = 2^{24} / 2^{10} = 2^{14} pages
(The number of complete page tables = 2^{22} / 2^{14} = 256)

PA size = 256 MB (2^{28} bits)
Number of bits needed for PFN in PTE = 28 – 10 = 18

10. An early version of UNIX was developed for small PDP-11 computers without memory management hardware. It held only one process in memory at a time; a swap was needed on each context switch. An early disk drive for such systems was the RL01. This drive took 55 mS to seek to access the data, and that data could be transferred at a rate of 512 KB/S. If each process was 32 KB in size and the quantum was 0.3 S, what fraction of the time was the system doing useful work?

Process transfer rate = (32 KB / 512 KB/s) + 55 ms = 0.1175 s

Fraction of the time the system did useful work = 0.1175 s / 0.3 s = 39.16%

12. Does using paged memory management hardware have any value even if we never swap pages to disk and back?

Paged memory management hardware separates the virtual address space from the physical address space. Processes can have a 64 bit virtual address space even though the system has less than 2^{64} bytes of physical memory.

16. Suppose that for a given job, Belady's min would result in a 99.99% hit rate, that memory access time is 70 nS, and that disk access time is 12 mS. If NFU results in 20% more page faults than Belady's min, and second chance results in 30% more page faults than Belady's min, how much is the average access time degraded for each of these page replacement policies?

Memory access time = 70 ns
Disk access time = 12 ms = 12000 us = 12000000 ns

Average access time for Belady's min = 99.99% * 70 ns + 0.01% * 12000000 ns
= 69.993 ns + 1200 ns = 1269.993 ns

Average access time for NFU = (1 – 0.01% * 120%) * 70 ns + (0.01% * 120%) * 12000000 ns
= (1 – 0.012%) * 70 ns + 0.012% * 12000000 ns
= 99.98% * 70 ns + 0.012% * 12000000 ns
= 69.986 + 1440 ns = 1509.986 ns

Average access time for Second Chance = (1 – 0.01% * 130%) * 70 ns
+ (0.01% * 130%) * 12000000 ns
= (1 – 0.01% * 130%) * 70 ns + (0.01% * 130%) * 12000000 ns
= (1 - 0.013%) * 70 ns + 0.013% * 12000000 ns
= 99.98% * 70 ns + 0.013% * 12000000 ns
= 69.986 + 1560 ns = 1629.986 ns

19. Most systems don't swap out pages that are part of a read-only text segment because they can be reread from the original executable file. Would there be any advantage to doing so?

The advantage to doing this prevents the need to have to access the secondary storage to swap out a page.

Chapter 11

1. Given our discussion of memory usage in Inferno, can Inferno support shared code between processes? Can multiple processes share the global data of a module?

Each of the modules used by a process has its own code that occupies the memory block where it was loaded, so Inferno doesn't support shared code between processes. On the other hand, each module has a global data block, which allows for multiple process to share the global data of a module.

4. When adding a block back to the free tree for a pool, we check to see if adjacent blocks are also free and coalesce them if they are. Is it possible that we might accidentally coalesce free blocks belonging to different pools in this way? Why or why not?

It is not possible that we might accidentally coalesce free blocks belonging to different pools in this way. This is because in *poolfree()* we are only looking at blocks in one pool, and blocks in a pool cannot belong to another one.