CS411/511. Operating Systems

Homework 3 - Solutions

Chapter 8: Review Questions 8.4, 8.10, 8.11, 8.16

511 students only: 8.17

8.4. When a process is rolled out of memory, it loses its ability to use the CPU (at least for a while). Describe another situation where a process loses its ability to use the CPU, but where the process does not get rolled out.

- When an interrupt occurs the process loses the CPU, but regains it as soon as the handler completes. The process is never rolled out of memory.

  Note that when timerrunout occurs, the process is returned to the ready queue, and it may later be rolled out of memory. When the process blocs, it is moved to a waiting queue, where it may also be rolled out at some point.

8.10. Consider a paging system with the page table stored in memory.

(a) If a memory reference takes 200 nanoseconds, how long does a paged memory reference take?

(b) If we add associative registers, and 75% of all page-table references are found in the associative registers, what is the effective memory reference time? (Assume that finding a page-table entry in the associative registers takes zero time, if the entry is there.)

- 400 nanoseconds. 200 ns to access the page table plus 200 ns to access the word in memory.
- 250 nanoseconds. 75% of the time it's 200 ns, and the other 25% of the time it's 400ns, so the equation is:

```
e.a. = (.75*200)+(.25*400)
```

Try this, too: What if the time to access the associative registers is actually 2 ns -- how does your answer change?

```
e.a. = 2 + (.75*200)+(.25*400)
```

Remember that you *always* have to perform the TLB lookup, whether or not the page is found there.

8.11. What is the effect of allowing two entires in a page table to point to the same page frame in memory? Explain how this effect could be used to decrease the amount of time needed to copy a large amount of memory from one place to another. What would the effect of updating some byte in the one page be on the other page?

- This allows users to shared code or data. If the code is reentrant, much memory space can be saved through the shared use of large programs (e.g., text editors, compilers, database systems). "Copying" large amounts of memory could be effected by having different page

tables point to the same memory location.

However, sharing of non-reentrant code or data means that any user having access to the code can modify it and these modifications would be reflected in the other user's "copy."

8.16. Consider the following segment table:

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are the physical addressed for the following logical addresses?

(a) 0,430
(b) 1,10
(c) 2,500
(d) 3,400
(e) 4,112

- (a) 219 + 430 = 649
- (b) 2300 + 10 = 2310
- (c) illegal reference; traps to operating system
- (d) 1327 + 400 = 1727
- (e) illegal reference; traps to operating system

8.17. Consider the Intel address translation scheme shown in Figure 8.28

(a) Describe all the steps that are taken by the Intel 80386 in translating a logical address into a physical address.
(b) What are the advantages to the OS of hardware that provides such complicated memory translation?
(c) Are there any disadvantages to this address translation system?

- (a) The selector is an index into the segment descriptor table. The segment descriptor result plus the original offset is used to produce a linear address with dir/page/offset. The dir is an index into a page directory; the entry from this directory selects the page table, and the page field is an index into that page table. The entry from the page table, plus the offset, is the physical address.

- (b) Such a page translation mechanism offers the flexibility to allow most OS's to implement their memory scheme in hardware, instead of having to implement some parts in hardware and some in software. Because it can be done in hardware, it is more efficient -- and the kernel is simpler.

- (c) Address translation can take longer due to the multiple table lookups it can involve. Caches help, but there will still be cache misses.

Chapter 9: Review Questions 9.3, 9.11, 9.18

511 students only: 9.5, 22.8

9.3. A certain computer provides its users with a virtual memory space of 2**32 bytes. The computer has 2**18 bytes of physical memory. The virtual memory is implemented by paging, and the page size is 4K bytes. A user process generated the virtual address 11123456. Explain how the system establishes the corresponding physical location. Distinguish between software and hardware operations.

- The virtual address in binary form is

  ```
  0001 0001 0001 0010 0011 0100 0101 0110
  ```

  Since the page size is 2**12, the page table size is 2**20. Therefore, the low-order 12 bits (0100 0101 0110) are used as the displacement into the page, while the remaining 20 bits (0001 0001 0001 0010 0011) are used as the displacement in the page table.

  Consider the operations that are needed (a) for DAT, and (b) for page fault servicing. All the DAT operations are carried out in hardware. But of the list of operations for page faults, on pp. 297-298, *at*least* steps 2, 4, 5, 6, 8, 10, and 12 involve software operations.

9.11. Consider the following page reference string:

```
1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6
```

How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, size, or seven frames? Remember that all frames are initially empty, so your first unique pages will all cost one fault each.

LRU replacement
FIFO replacement
Optimal replacement

# Frames	LRU	FIFO	Optimal
1	20	20	20
2	18	18	15
3	15	16	11
4	10	14	8
5	8	10	7
6	7	10	7
7	7	7	7

9.17 (not in the assignement, but I'll leave the solution here).
Consider a demand-paging system with a paging disk that has an average access and transfer time of 20 ms. Addresses are translated through a page table in main memory, with an access time of 1 us per memory access. Thus, each memory reference through the page table takes two accesses. To improve this time, we have added an associative memory that reduces access time to one memory reference, if the page-table entry is in the associative memory.

Assume that 80% of the accesses are in the associative memory, and that, of the remaining, 10% (or 2% of the total) cause page faults. What is the effective memory access time?

```
e.a. = 1 us + (0.20 * 1 us) + (0.02 * 20,000 us)
     = 401.2 us
```

Or, if you prefer

```
e.a. = (0.80 * 1 us) + (0.18 * 2 us) + (0.02 * 20,002 us)
     = .8 us + .36 us + 400.04 us
     = 401.2 us
```

9.18. Consider a demand-paged computer system where the degree of multi-programming is currently fixed at four. The system was recently measureed to determine utilization of CPU and the paging disk. The results are one of the following alternatives.

For each case, what is happening? Can the degree of multiprogramming be increased to increase the CPU utilization? Is the paging helping?

a. CPU utilization 13 percent; disk utilization 97 percent

System is thrashing. The degree of multiprogramming should be decreased. Paging is not helping.

b. CPU utilization 87 percent; disk utilization 3 percent

System is well utilized, CPU is being kept busy most of the time. The degree of multiprogramming probably should stay the same, increasing it may lead to thrashing. Paging is helping.

c. CPU utilization 13 percent; disk utilization 3 percent

System is under utilized, the CPU is not getting enough work. The degree of multiprogramming should be increased. Paging is not really helping or hurting.

9.5. Suppose we have a demand-paged memory. The page table is held in registers. It takes 8 ms to service a page fault if an empty page is available or the replaced page is not modified, and 20 m if the replaced page is modified. Memory access time is 100 ns.

Assume that the page to be replaced is modified 70% of the time. What is the maximum acceptable page-fault rate for an effective access time of no more than 200 ns?

```
[note:  everything shown in microseconds]
0.2 us = ((1-P) * 0.1 us) + (0.3P * 8,000 us) + (0.7P * 20,000 us)
   0.1 = -01.P + 2,400P + 14,000P
   0.1 ~ 16,400 P
     P ~ 0.000006
```

22.8. What are three advantages of dynamic (shared) linkage of libraries, compared to static linkage? What are two cases where static linkage is preferable?

- The primary advantages are that they reduce the memory and disk space used by a system, and they enhance maintainability. (You should be able to describe in detail why this is so.) Other advantages are the fact that a program that uses shared libraries can often be adapted

for other purposes simply by replacing one or more of its libraries (e.g., substituting a debugging library for the normal one in order to trace a problem in an application). Shared libraries also allow program binaries to be linked against commercial, proprietary library code without actually including any of that code in the program's final executable file -- so it's may be possible to distribute code to a third party without having to incur extra licensing charges.

- Static linkage is more appropriate for "system administration rescue" situations, such as if a sysadmin makes a mistake while installing a new library that causes it to be corrupted. Therefore, it is common to provide a set of basic "rescue utilities" that are statically linked, so that the fault can be corrected without having to rely on shared libraries. Since dynamic linking adds to the execution time, there may be performance reasons for linking the libraries statically. Also, it's easier to distribute an executable file that is complete (i.e., statically linked) rather than having to count on the recipient having the appropriate shared librares.
