Name: ____________________________

Operating Systems
V22.0202 Spring 2010

Midterm Exam
ANSWERS

1. **True/False (10 points)**. Circle the appropriate choice.

   (a) **T** A process is a program in some state of execution.

   (b) **F** A physical address indicates a block on the disk where data can be found.

   (c) **T** The memory bus is a set of wires running between the CPU and main memory.

   (d) **F** A controller is the portion of the OS responsible for communicating with an I/O device.

   (e) **F** The Not Frequently Used (NFU) page replacement algorithm keeps a precise count of the number of times that a page has been referenced.

   (f) **T** In a system where lottery process scheduling is used, efficient use of the CPU can be increased by allocating more "lottery tickets" to I/O-bound processes than to compute-bound processes.

   (g) **F** If all the processes on a computer are blocked, the system must be rebooted because there is no way for any of the blocked processes to become ready.

   (h) **T** Upon a TLB miss, a system that uses a 4-level page table will incur more memory references than a system that uses a single-level page table.

   (i) **F** In round-robin scheduling, a longer quantum means that an interrupt from the disk controller will take longer to be handled by the OS.

   (j) **F** The threads within a process have their own registers, but use the same code, global data, and stack.

2. **(5 points each) Fill in the blanks provided on this sheet.**

   (a) Suppose that on a machine with 16-bit addresses and 512 byte pages, the TLB contains the following:

1	0	1	0C	38
1	1	1	45	1D
1	0	0	19	22
0	1	1	37	66

   where the fields are, from left to right, the valid bit, the R bit, the M bit, the virtual page and the page frame. The numbers are given in hex. What is the physical address corresponding to the virtual address 194A? If that information cannot be determined from the TLB, indicate why. (NOTE: Consider the individual bits carefully)

   Since the page size is 512 bytes, the lowest 9 bits of an address is used as the offset within a page. Thus, the remaining 7 bits of the 16-bit address are used for the page number. The virtual address 194A hex, given above,

corresponds to the bits 0001 1001 0100 1010. Thus, the upper 7 bits are 0001100, which is 0C in hex. The top entry of the given TLB maps page OC to page frame 38. Concatenating the seven page frame bits representing 38 hex, namely 0111000 to the lower 9 bits of the above virtual address results in 0111 0001 0100 1010, which is 714A hex.

Answer: 714A hex

(b) Suppose the TLB uses the NRU algorithm to evict an entry, when a page has been referenced for which there is no TLB entry. What is the first entry in the above TLB (from part (a)) that will be evicted? What is the second TLB entry to be evicted?

The first entry of the TLB to be overwritten is the empty one, namely the one with the valid bit equal to zero. That would be the bottom entry of the above TLB. According to NRU, if there are no empty entries, then pick the entry to evict that has a zero in both the R bit and the M bit. That would be the entry that is second from the bottom in the above TLB.

Answer: First: Bottom entry   Second: Second from bottom entry

3. (5 points each part) Put your answer in the blue book.

This question is about how *you* would implement semaphores in *your* first programming project (the interrupt handler & scheduler assigment).

Suppose the first programming assignment had specified that you had to implement binary semaphores as follows:

- The OS supports 50 binary semaphores, which are numbered 0 through 49 and are referenced by number, not by name.
- There is a trap called `SEM_DOWN` that, when invoked by a running process, causes the number of the semaphore being used to be put in the R2 register.
- There is a trap called `SEM_UP` that, when invoked by a running process, causes the number of the semaphore being used to be put in the R2 register.

(a) How would you represent the 50 semaphores in your kernel.c code?

Each semaphore should have a value (0 or 1) and a queue of PIDs of the processes that are blocked on that semaphore. Thus, each semaphore could be represented by a struct that contained an integer value and a queue. The 50 semaphores would be represented by an array of 50 of these structs.

(b) Describe precisely, in C code or pseudo-code, how your trap handler procedure would handle the SEM_DOWN trap.

The trap handler would handle the SEM_DOWN trap as follows:

- **If semaphores[R2].value == 1 (i.e. the value field of the R2th element of the semaphores array is 1), then set semaphores[R2].value = 0 and return. The current process will continue to execute.**
- **Else, if semaphores[R2].value == 0, the current process needs to block. Set the state of the current process to BLOCKED, add the current process to the end of the queue of processes that are blocked on semaphores[R2],**

and call the round-robin scheduler to pick a new process to run from the ready queue.

- **Otherwise, if semaphores[R2].value is some value other than 0 or 1, this is an error condition. The program can either print an error and exit or handle the error in some other way (you didn't have to consider this case).**

(c) Describe precisely, in C code or pseudo-code, how your trap handler procedure would handle the SEM_UP trap.

The trap handler would handle SEM_UP as follows:

- **If semaphores[R2].value == 0 and the queue of processes blocked on semaphores[R2] is empty, then set semaphores[R2].value to 1 and return. The current process continues to execute.**
- **Else, if semaphores[R2].value == 0 and the queue of processes blocked on semaphores[R2] is not empty, do not modify semaphores[R2].value. Instead, remove a process from the head of the queue of processes blocked on semaphores[R2], set the state of that process to READY, put that process on the ready queue, and return. The current process continues to execute.**
- **Otherwise, if semaphore[R2].value is not 0, this is an error condition.The program can either print an error and exit or handle the error in some other way (you didn't have to consider this case).**

4. (5 points each part) Put your answer in the blue book.

On a machine with 32-bit addresses, suppose an OS uses two-level page tables as follows:

- A first level page table has 2048 entries.
- Each second level page table has 2048 entries.

(a) What is the size of a page?

The index into the first page table requires log(2048) = 11 bits of the 32-bit address, as does the index into the second page table. Thus, the remaining bits, 32 - (11 + 11) = 10, are the offset into the page. The page size is 2^{10} = 1024 = 1K bytes.

(b) Suppose a process has a text segment that occupies 64MB, a stack segment that occupies 128MB, and a data segment that occupies 32MB. How much space is occupied by the page tables for that process? Your answer can be given in powers of 2, if you want, to avoid arithmetic.

As stated in class and written on the blackboard during the exam, you were to assume that each page table entry is 4 bytes. Since the first page table has $2048 = 2^{11}$ entries, it occupies $2^{11} * 4 = 2^{13}$ bytes.

Only the second-level page tables that are needed for the actual text, data, and stack segments are allocated. Since each page is 2^{10} bytes, the number of pages needed to hold the text segment is $64\text{MB}/2^{10} = 2^{26}/2^{10} = 2^{16}$. Similarly, the number of pages needed to hold the stack segment is $128\text{MB}/2^{10} = 2^{27}/2^{10} = 2^{17}$ and the number of pages to needed hold the data segment is $32\text{MB}/2^{10} = 2^{25}/2^{10} = 2^{15}$.

Since each second-level page table has $2048 = 2^{11}$ entries (with one entry per page in the process), the number of second-level page tables needed for the text segment is $2^{16}/2^{11} = 2^5 = 32$. Similarly, the number of second-level page tables needed for the stack segment is $2^{17}/2^{11} = 2^6 = 64$ and for the data segment is $2^{15}/2^{11} = 2^4 = 16$. Thus, the total number of second-level page tables is $32 + 64 + 16 = 112$. Since each second-level page table contains 2048 entries of 4 bytes each, it occupies $2^{11}*4 = 2^{13}$ bytes. Thus, all the second-level page tables occupy $112 * 2^{13}$ bytes.

Since the first page table also occupies 2^{13} bytes, the total amount of space occupied by all the page tables is $(112 + 1) * 2^{13} = 113 * 2^{13}$. To simplify (which you didn't have to), this is $113 * 8K = 904$K bytes.

(c) What does each entry in the first-level page table contain?

A pointer to a second-level page table (or null).

(d) In a two-level page table mechanism, what does each TLB entry contain? How is it different than a TLB entry on a system with a single-level page table mechanism?

bf The structure of a TLB entry is the same for single-level page tables as for multi-level page tables. It contains a valid bit, R bit, M bit,page number, and page frame number and possibly other bits for protection, etc.

(e) When a process references an address, precisely how does the MMU determine if the page containing the referenced address is in RAM and, if so, in which page frame. Be precise regarding 1) how the bits of the address are used and 2) the steps that the MMU goes through.

First, the MMU checks if the page number, which is the upper 22 bits of the address in the above system, is found in a valid entry of the TLB. If so, the corresponding 22-bit page frame in the TLB is concatented with the lower 10 bits of the address to form the 32-bit physical address. If no valid entry containing the page number is found in the TLB, then the top 11 bits of the address are used as an index into the first-level page table. If the entry in the first-level page table is a valid pointer to a second level page table, then the next 11 bits of the address are used to index into that second level page table. If the second-level page table entry at that index has its present bit set to 1, then the 22-bit page frame found in the entry is used. Otherwise, a page fault occurs.

(f) What happens if it turns out that the page containing the referenced address is not in memory?

The MMU issues a page fault, which is an interrupt handled by the OS. The OS will retrieve the requested page from the disk and write it into a page frame in memory. If necessary, a page residing in memory will need to be evicted to make room for the incoming page.