Operating Systems
V22.0202 Fall 2008

Midterm Exam

Answers

1. **True/False**. Circle the appropriate choice.

   (a) **T** In priority scheduling, I/O-bound processes should be given a higher priority than CPU-bound processes in order to make more efficient use of the CPU.

   (b) **F** All computers must have at least two CPUs, since the operating system needs to execute at the same time as user programs.

   (c) **T** A short quantum in a round-robin scheduler gives better response time for interactive users but less efficient use of the CPU than a longer quantum.

   (d) **F** In a 2GHz Pentium, the clock interrupt occurs 2 billion times a second.

   (e) **F** The main difference between the use of test&set and the use of semaphores is that semaphores require the OS to do the busy waiting rather than the user program.

   (f) **F** Within a given virtual memory system, pages of varying sizes are used to avoid wasting space within partially-filled pages.

   (g) **F** For a machine with 32-bit addresses, the use of a two-level page table allows for a larger virtual address space than a single-level page table.

   (h) **F** A process is just the compiled version of a program.

   (i) **T** A processor is generally put into kernel mode through the execution of a trap instruction. **I will accept F too, since interrupts also put the processor kernel mode.**

   (j) **T** The purpose of an operating system is to manage system resources and to provide a more convenient abstract machine for programming.

2. **Multiple Choice.** Circle the correct answer.

   (a) If addresses are 27 bits, the size of the space that can be addressed is:

   - 64KB
   - **128MB**
   - 256MB
   - 4GB

   (b) If a 64TB (where TB=terabyte) address space is desired, then addresses must be at least:

   - 32 bits
   - 40 bits
   - **46 bits**
   - 64 bits

3. **Virtual Memory**. Put your (short!) answers in the blue book

(a) Suppose, on a machine with 16-bit virtual and physical addresses and a page size of 256 bytes, a process is running and the TLB contains the following:

1	32	4F
1	1A	C3
1	89	22
0	42	B2

where, from left to right, the columns contain the valid bit, the virtual page number, and the page frame number. All numbers are given in hexadecimal. If the process issues the virtual address 1AF2 hex, what physical address (in hex) will the MMU issue? Show your work, but without giving a long explanation.

Since a page size is 256 bytes, an offset into a page is 8 bits. That leaves 8 bits of a virtual address to identify the page number. Thus, a virtual address is partitioned as follows:

8-bit page number	**8-bit offset**

Since 8 bits is two hex digits, when looking up the virtual page number for the address 1AF2 in the TLB, the top two hex digits, 1A, are used. In the above TLB, the corresponding page frame number is C3, so the resulting physical address, comprised of the 8-bit page frame number and the original 8-bit offset, would be C3F2.

(b) Why are multi-level page tables often used instead of ordinary (single-level) page tables? What is the added cost associated with using multi-level page tables?

For most processes, the number of pages that are actually allocated to store the code, data, and stack for the process is far less than the maximum number of pages contained in a virtual address space. In a single level page table, the number of entries would be the number of possible pages. On the other hand, in a two-level page table, for example, only a sufficient number of second level page tables are required in order to contain entries for pages that are actually allocated. Thus, there is a substantial savings in the space occupied by a two-level page table over a single-level page table. In a three-level page table (for very large address spaces), the space savings is even greater.

The added cost of a multi-level page table is the additional memory reference(s) needed to access the first level page table, then a second level page table, etc. This additional overhead is reduced substantially, though, by the use of a TLB.

(c) Suppose there is a machine with 32-bit addresses and a two-level page table (in memory) such that the first 10 bits of an address is an index into the first level page table and the next 10 bits are an index into a second level page table. Suppose also that each entry

in the page tables is 32-bits. How much space is occupied in memory by the page tables for a process that has 64MB of actual virtual address space allocated. Show your work without giving a long explanation.

As specified above, a 32-bit virtual address is partitioned as follows:

10-bit index in 1st-level PT	10-bit index in 2nd-level PT	12-bit offset

Since an index into the first level or second level page table is 10 bits, there must be 1K entries (i.e. 2^{10}) per page table. Since each page table entry is 4 bytes, a page table (either first or second level) occupies 4KB.

Because 12 bits are used to offset into a page, a page must be 4KB (i.e 2^{12}) in size.

Since the process uses 64MB (i.e. 2^{26} bytes) and each page is 4KB (i.e. 2^{12} bytes), there must be 2^{14} pages. Thus, we need a sufficient number of second level page tables to hold 2^{14} entries. Since each second level page table has 2^{10} entries, we need 16 (i.e. 2^4) second level page tables.Therefore, because there is one first-level page table and 16 second level page tables, the total space occupied in memory by the page tables is (17 * 4KB) = 68KB.

4. **Semaphores**. Put your answers in the blue book.

   Suppose you were the implementor of an operating system that provided support for semaphores.

   (a) Describe how you would implement the down(S) and up(S) systems calls, where S is a semaphore.

   The down(S) system call would trap to the OS. The trap handler for the OS would check the status of the semaphore S. If the semaphore is already 0, then the OS would block the process and invoke the scheduler to choose a ready process to run. If the semaphore was not 0, then the semaphore is decremented and the the scheduler is invoked to choose a process to run (it may be the process that performed the down(S)).

   The up(S) system call would also trap to the OS. The trap handler for the OS would check if there were any processes that were blocked on the semaphore S. If so, it would choose one of the blocked processes, set its state to ready and then invoke the scheduler to choose a process to run next (it may still be the process that performed the up(S)). If there are no waiting processes, then the value of the semaphore is incremented.

   (b) How would you represent the semaphore S itself?

   The value of the semaphore could be represented as an integer in memory. Along with the integer, a list of processes that are blocked on that semaphore would need to be stored. Thus, a semaphore could be represented as a pair: an integer and a pointer to the head of the list of blocked processes.

   (c) How would you keep track of the processes that are blocked because they performed a down(S)?

   Each time a process performs a down(S) when S is already 0, the process (actually, just its pid) gets put on the linked list of processes blocked on S.

When a process performs an up(S), a blocked process on that list is removed and added to the ready queue.

If it helps, you can give your description in terms of the simulated system from the programming project. Be sure to account for everything that happens when a down() or up() operation is performed.