# CSE 410 Final Exam 6/09/09

Name ________________________________

Do **not** write your id number or any other confidential information on this page.

There are 12 questions worth a total of 100 points. Please budget your time so you get to all of the questions. Keep your answers brief and to the point.

You may want to use a copy of the "green card" from the textbook. We have additional copies if you did not bring one with you. You may also have a sheet of hand-written notes if you brought them, as well as the sheet of notes you had for the midterm. Other than that, the exam is closed book, closed notes, closed calculators, closed laptops, closed twitter, closed telepathy, etc.

Please wait to turn the page until everyone is told to begin.

Score __________________ / 100

| | |
|---|---|
| 1. | _______ / 10 |
| 2. | _______ / 6 |
| 3. | _______ / 8 |
| 4. | _______ / 8 |
| 5. | _______ / 8 |
| 6. | _______ / 8 |
| 7. | _______ / 8 |
| 8. | _______ / 10 |
| 9. | _______ / 8 |
| 10. | _______ / 10 |
| 11. | _______ / 8 |
| 12. | _______ / 8 |

**Question 1.** (10 points) (Caches)

Suppose we have a memory and a direct-mapped cache with the following characteristics.

- Memory is byte addressable
- Memory addresses are 16 bits (i.e., the total memory size is $2^{16}$ = 65536 bytes)
- The cache has 8 rows (i.e., 8 cache lines)
- Each cache row (line) holds 16 bytes of data

(a) In the spaces below, indicate how the 16 address bits are allocated to the offset, index, and tag parts of the address used to reference the cache:

_________ offset bits

_________ tag bits

_________ index bits

(b) Below is a sequence of four binary memory addresses in the order they are used to reference memory. Assume that the cache is initially empty. For each reference, write down the tag and index bits and circle either hit or miss to indicate whether that reference is a hit or a miss.

| Memory address | Tag | Index | Hit / Miss (circle) |
|---|---|---|---|
| 0010 1101 1011 0011 | | | Hit Miss |
| 0000 0110 1111 1100 | | | Hit Miss |
| 0010 1101 1011 1000 | | | Hit Miss |
| 1010 1010 1010 1011 | | | Hit Miss |

**Question 2.** (6 points) (hardware) Different storage devices have different access times and costs. For each of the following devices, circle the access time that is closest to the typical time needed to read information from that device given current hardware technologies. The important thing is to get the order of magnitude right; the exact number is not the issue. (ms = millisecond, µs = microsecond, ns = nanosecond). If it matters, assume that these numbers are for a computer with a 1GHz clock (i.e., a low-end current machine)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Main memory (RAM): | 1sec | 100ms | 10ms | 1ms | 100µs | 10µs | 1µs | 100ns | 10ns | 1ns |
| Hard disk drive: | 1sec | 100ms | 10ms | 1ms | 100µs | 10µs | 1µs | 100ns | 10ns | 1ns |
| CPU register: | 1sec | 100ms | 10ms | 1ms | 100µs | 10µs | 1µs | 100ns | 10ns | 1ns |

**Question 3.** (8 points) (cache organization) Two of the design choices in a cache are the row size (number of bytes per row or line) and whether each row is organized as a single block of data (direct mapped cache) or as more than one block (2-way or 4-way set associative).

The goal of a cache is to reduce overall memory access time. Suppose that we are designing a cache and we have a choice between a direct-mapped cache where each row has a single 64-byte block of data, or a 2-way set associative cache where each row has two 32-byte blocks of data. Which one would you choose and why? Give a brief technical justification for your answer. If the choice would make no difference in performance then explain why not.

**Question 4.** (8 points) (threads and processes) In a system with both processes and threads, context switches occur both between threads in the same process and between processes. Below is a list of items that are often stored in Thread Control Blocks (TCBs) or Process Control Blocks (PCBs) or both. For each item, if the item is loaded or saved to/from a processor register during a process or thread context switch, put an X or a check mark in the respective column. If it is not loaded or saved, leave the column blank.

| Item | Saved/restored during process context switch | Saved/restored during thread context switch |
|---|---|---|
| Program counter register (PC) | | |
| Stack pointer register (SP) | | |
| General purpose registers | | |
| User name of owner | | |
| Scheduling priority | | |
| Processor register containing address of page table | | |
| Information about open files | | |

**Question 5.** (8 points) (threads) A common model for implementing threads is a user-level thread package where a single kernel thread is shared among all the threads in a user address space. Using this model, if the running user thread performs an I/O operation that blocks the kernel thread, no other user thread can run, even if other user threads have work available that they could do. Give two different ways that this model can be changed or extended to allow other user threads in the same address space to continue executing even if one thread performs a blocking I/O operation.

**Question 6.** (8 points) (synchronization) Both semaphores and monitors provide a method of synchronizing concurrent access to data shared by multiple threads. Why would anyone want to use monitors instead of semaphores? What advantage(s) is (are) there to doing this?

**Question 7.** (8 points) (process scheduling) In most priority schedulers, a thread's priority is increased and decreased dynamically over time.

(a) Why would a typical scheduler increase the priority of an executing thread? What problem(s) would be solved by doing this?

(b) Why would a typical scheduler decrease the priority of an executing thread? What problems(s) would be solved by doing this?

**Question 8.** (10 points) (synchronization) Your friend Ben Bitblitter is trying to implement locks to use with concurrent processes. In Bitblitter's implementation, a lock is a variable whose value is 0 if the lock is free and is the non-zero process id number of the process that holds the lock if it is in use. Bitblitter proposes to implement the lock operations as follows (using the pseudo-notation from the lecture slides, which doesn't necessarily match any specific programming language, but is similar to C or Java).

```
struct lock {         /* lock data structure:           */
  int owner = 0;      /*    lock owner process id or 0 */
}                     /*    if not locked              */

void acquire(lock) {
  pid = get_process_id();       /* get process id */
  while (lock->owner != 0) { }  /* busy wait */
  lock->owner = pid;
}

void release(lock) {
  lock->owner = 0;
}
```

Bitblitter's implementation is intended to run on multiprocessor machines, so there is no problem using a busy wait in the implementation of `acquire`. But even so, his code doesn't always work. Most of the time it behaves properly, but every now and then something goes wrong.

(a) What is the bug in the code? What can go wrong? Your answer does not need to be long, but it does need to be precise.

(b) What needs to be changed for this code to work properly? How might we do this? (Recall that this code is intended to work on a multiprocessor system, if that makes any difference.)

**Question 9.** (8 points) (concurrency) Three painters are working on repainting an old house. They all are working part-time, so often only one or two of them are around.

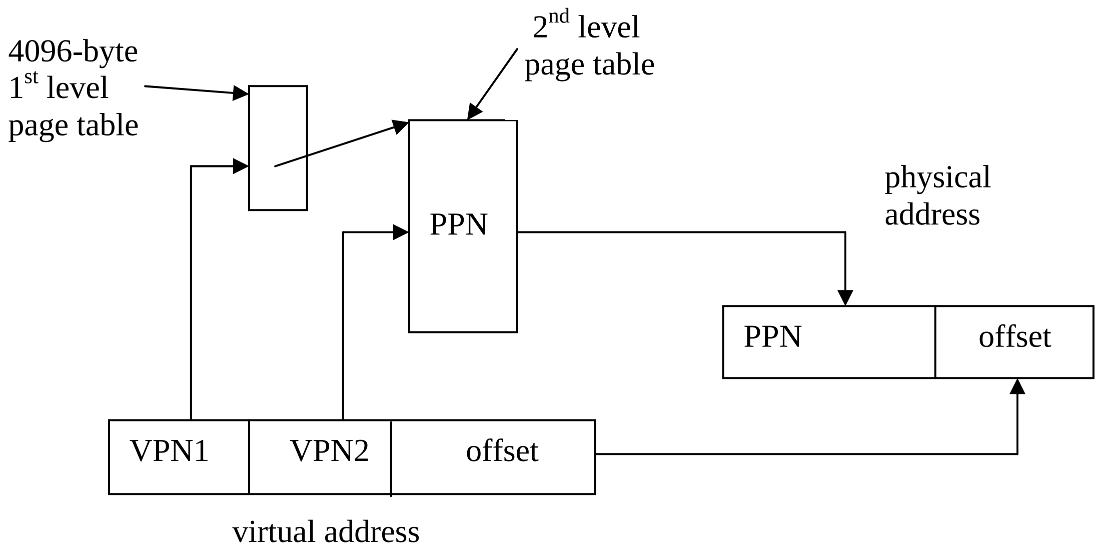
Because of budget cuts, only two buckets of paint and two paintbrushes are available. When a painter shows up to work, he grabs one of each and starts painting. However, if he can only grab a bucket or a brush but not the other, or if neither is available, he grabs what he can get and then waits around until one of the painters who is working finishes and frees up a brush and a bucket. Painters are stubborn and once they have either a bucket or a brush they will wait indefinitely for the other to become available so they can paint. But once a painter gets both a brush and a bucket, he will start working and continue until he is done, without worrying about what the other painters are doing.

Is it possible for deadlock to occur in this situation, where at least one painter is waiting indefinitely for either a bucket or a brush or both? Describe how this could happen, or explain why it will never occur.

**Question 10.** (10 points) (VM) To avoid excessively large page tables, many virtual memory systems have multi-level page tables. For this problem, we would like to figure out the details of a 2-level paging system with the following characteristics:

- Both virtual and real (physical) memory addresses are 32 bits each
- Pages are 4096 bytes each
- Page table entries are 4 bytes each, containing the physical page frame number, plus additional control bits including the valid bit.
- The top-level page table consists of a single 4096-byte page and has as many page table entries as will fit.

The page table structure and address translation can be diagramed like this (similar to the multi-level page table diagram in the slides).



Fill in the blanks with the correct numbers to match the given constraints:

Number of entries in the first level page table ________________

Number of bits in the VPN1 part of the virtual address ______________

Number of bits in the offset field of each address ________________

Number of bits in the VPN2 part of the virtual address ______________

Number of page table entries in each 2nd level page table _____________

**Question 11.** (8 points) (VM and caches) Most memory systems contain both a translation lookaside buffer (TLB) used by the virtual memory system, and a cache memory. Both the cache and the TLB have similar functions: they store main memory data in a faster "cache-like" memory that can be accessed at nearly processor speeds.

The question is why do these two separate mechanisms exist when they have such a similar purpose? Why have both a cache and a TLB? Why not just a single "super-cache" that would hold both regular memory data as well as frequently used page table entries?

Hint: What is different about these two fast memories, if anything? How are they used in the memory system?

**Question 12.** (8 points) (file systems) The classic Unix file system stores information about files in various places: the file system superblock, file i-nodes, and data blocks that contain file and directory data.

For this question, place an X in the correct column showing where each of the given pieces of information are stored in a classic Unix file system.

| What | Superblock | i-node | directory data block | file data block |
|---|---|---|---|---|
| File name | | | | |
| File owner name | | | | |
| File permissions (owner, group, others) | | | | |
| Disk location of first data block | | | | |
| Disk location of start of i-node section on the disk | | | | |
| File creation date | | | | |
| File data (text files, databases, etc.) | | | | |
| File link count (number of directory links to a file) | | | | |