CSE 410 Final Exam 6/08/10

Name ________________________________

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

There are 10 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. 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. ______ / 12
2. ______ / 12
3. ______ / 12
4. ______ / 10
5. ______ / 8
6. ______ / 10
7. ______ / 10
8. ______ / 10
9. ______ / 8
10. ______ / 8

Question 1. (12 points) (caches)

(a) One choice in designing cache memories is to pick a block size. Which of the following do you think would be the most reasonable size for cache blocks on a computer designed for general-purpose laptop or desktop applications? Circle your answer and justify it.

8 bytes 64 bytes 512 bytes 2048 bytes

(b) What problem does a 2-way associative cache solve compared to a direct-mapped one?

(c) A design choice for a cache memory is whether to use a write-back or a write-through policy. What is the difference between these?

Question 2. (12 points) Suppose we have a memory system that has a main memory, a single-level cache, and paging virtual memory. The three levels of the memory system have the following access times:

2 ns	cache
100 ns	main memory
10 ms	paging disk

(a) The cache has a 95% hit rate. What is the effective memory access time if we consider only the cache and main memory and ignore page faults and disk access times? (To save time, you only need to clearly write out the formula, but do not have to simplify the final result.)

(b) Now recalculate the effective memory access time assuming the same cache hit rate (95%) plus a page fault rate of 0.001% (i.e., 99.999% of the memory accesses succeed without producing a page fault). (Again, a formula is adequate.)

Question 3. (12 points) (paging) (a) Belady's MIN algorithm for replacing pages in a virtual memory system has the provably lowest page-fault rate of all theoretical or practical page replacement algorithms. What is this page replacement policy?

(b) No real systems use Belady's MIN algorithm. Why not?

(c) There are several other page replacement algorithms that could be used instead of MIN in a real system including FIFO, approximate LRU, and randomly picking a page frame to reuse after a page fault. Which one of these is the best choice for realistic systems and why?

Question 4. (10 points) (threads) Threads can be implemented as user-level threads using a library package that runs entirely in a process address space, or as kernel-level threads, where the operating system kernel is aware when a process has more than one thread associated with it.

(a) Give one advantage that user-level threads have compared to kernel-level threads.

(b) Give one advantage that kernel-level threads have compared to user-level threads.

Question 5. (8 points) Translation Lookaside Buffers (TLBs) have often been implemented with a fully associative lookup, while cache memories never use this. Give a brief technical explanation of why a fully associative lookup scheme can make sense for a TLB but not for a cache.

Question 6. (10 points) (scheduling) Suppose we have the following jobs to execute, with the given start times and total times needed for each job.

Job	Start Time	Total time
A	0	25
B	10	60
C	25	30

Now suppose we execute those jobs using a Round-Robin scheduler with a time quantum of 20. Fill in the chart below to show the timeline of when each job executes, starting with time 0 in the first row, and filling in each successive row with a time and the letter identifying the job that starts or resumes execution at that time. Continue until all of the jobs are finished. You may not need all the rows. The first row is filled in for you

Time	**Job**
0	A

Question 7. (10 points) (locks) In lecture we discussed the implementation of low-level lock primitives `acquire` and `free`. If we represent a lock as a word in memory with the value 1 when it is held by a thread and 0 when it is free, the `free` operation is easy – just store 0 in the lock variable. Acquire needs to set the lock variable to 1, but it must be done correctly in case some other thread holds the lock or is trying to acquire it.

A common instruction provided on many computers to help with this is compare-and-swap (`cas`). A typical version of this involves three registers – two containing values and a third containing the address of the lock. Here is a pseudo-code description of the operation of `cas`:

```
cas  x,y,addr ≡
   temp = MEM[addr];
   if (temp == x) {
      MEM[addr] = y;
   } else {
      x = temp;
   }
```

(a) Complete the following assembly language implementation of the operation `acquire` that so it can be executed by a thread to acquire a lock correctly. Register `$a0` contains the address of the lock variable. Fill in the blanks, including the correct conditional branch operand on the "b____" instruction.

```
# acquire the lock whose address is given in $a0.

acquire:

     li      $t0, ________

     li      $t1, ________

     cas     $t0, $t1, $a0

     b____   ______ , ______ , acquire

                  # repeat if lock not acquired yet

     jr      $ra  # return to caller once lock acquired
```

(continued next page)

Question 7. (cont.) Definition of compare-and-swap repeated here for reference:

```
cas  x,y,addr ≡
  temp = MEM[addr];
  if (temp == x) {
    MEM[addr] = y;
  } else {
    x = temp;
  }
```

(b) Compre-and-swap is always implemented as a special instruction in the processor and memory system. Give a technical reason why this is necessary and why the `cas` instruction cannot be correctly replaced by a software procedure that performs the same operations as shown in the pseudo-code above.

Question 8. (10 points) In languages like C, C++, Fortran, and others, it is possible to create an array of objects (structs) without allocating the objects individually with `new` or the equivalent construct. For example, suppose a single complex number is represented as a pair of `double`s.

```
class Complex {         // complex number
  double re;            // real part
  double im;            // imaginary part
}
```

When we create an array of `Complex` data, it would look something like this in memory:

0	1	2	3	4	5	6	7	8	9	10	…
re,im	re,im	re,im	re,im	re,im	…						

Now for the question. We would like to initialize this array to (0.0, 0.0). Here are two possibilities:

I

```
for (k = 0; k < n; k++) {
  a[k].re = 0.0;
}
for (k = 0; k < n; k++) {
  a[k].im = 0.0;
}
```

II

```
for (k = 0; k < n; k++) {
  a[k].re = 0.0;
  a[k].im = 0.0;
}
```

Is there any reason to prefer one of these solutions over the other for performance or other technical reasons? If so, why; if not, why not?

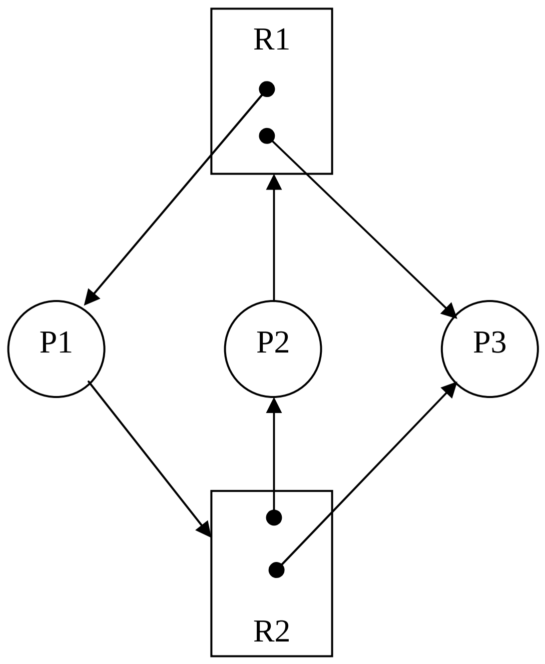
Question 9. (8 points) (synchronization) Inspired by our success in solving the bridge traffic problems (hw8), we have been invited back to solve a synchronization problem involving cows on a Vermont farm.

This farm has a milking barn with 3 stalls. Only 3 cows can fit in the barn at a time, however if another cow sticks its head in the door, it won't leave since it wants to be milked and eat its dinner.

Use a counting semaphore to implement a pair of procedures to control entry to the barn so that at most 3 cows are in the barn at any one time. Each cow will execute procedure `enter_barn()` before entering the barn and `exit_barn()` when leaving. You need to provide an initial value for the semaphore `ok_to_enter` and implementations of these two procedures below.

```
semaphore ok_to_enter = ___________ ;

void enter_barn() {



}

void exit_barn() {



}
```

Question 10. (8 points) (deadlocks) Consider the following resource allocation graph. Rectangles indicate resources; circles indicate processes. There is an arrow from a process to each resource it is requesting but does not hold, and an arrow from each resource to a process that holds a lock on that resource.



Is this system deadlocked? Why or why not? (Give a precise reason why there is a deadlock or how you can prove that the system is not deadlocked.)