CS-537: Midterm (Fall 2012)

This midterm is about Batman

Please Read All Questions Carefully!

There are SEVEN (7) total *numbered* pages.

Please put your FULL NAME (mandatory) on THIS page only.

Name: _____

Bruce Wayne is a wealthy businessman from Gotham City, who occasionally dresses up like a bat and beats up on criminals. To keep track of his crime fighting exploits, he decided to design his own operating system, BatOS, because billionaire playboy industrialist philanthropists don't use consumer-level operating systems.

Unfortunately, Bruce Wayne failed CS 537.

Mr Wayne's butler, Alfred, has hired you to talk to Mr Wayne about his initial design and correct his mistakes.

Remember to read all the questions carefully. Good luck!

1. **Scheduling.** Mr Wayne remembers that the name of one of the best policies for job scheduling is "shortest time to completion first" (STCF), and he wants to use this policy in BatOS. Describe this policy to Mr Wayne, and explain why he can't actually use it.

2. **Scheduling.** Why not a "round robin" (RR) policy, in honor of his sidekick? Mr Wayne would like an RR policy with 5 second time slices, so that he can have a good long time with the CPU when he's running a job. However, he wants to run 50 jobs simultaneously, all of which take 20 seconds to complete. What would his average turnaround time be? Average response time?

   Mr Wayne proposes shortening the time slice to one second. What are the average turnaround and response times now?

3. **Scheduling.** Of those 50 simultaneous jobs, most of them are batch jobs and only a few are interactive. You propose a multi-level feedback queue (MLFQ). Mr Wayne swears none of his batch jobs ever become interactive, and suggests removing the periodic queue level reset from the MLFQ design. Describe the life of a batch job in this scheduler. What is a potential problem?

4. **Scheduling.** "Too complicated!" says Mr Wayne. "Let's just give every job a number and use a random number generator to pick something to run." In typical lottery scheduling, every process gets a user-determined number of random number "tickets" and one is randomly selected every time slice. What is the effect of giving *every* job only *one* random number? Does this make the scheduler fair?

5. **Memory Virtualization.** Mr Wayne proposes segmentation for address translation in BatOS, but he remembers that the "grows positively" register that you add to the base/bounds registers for each logical segment made the math confusing. What purpose does that register serve? What is the effect if you don't have it?

6. **Memory Virtualization.** You remind Mr Wayne about paging, and that you can make address translation faster with a TLB on the CPU. "Ah yes!" he cries. "All I need in a TLB entry is the VPN, PFN, and valid bit, right?" What crucial part of a TLB entry is he missing? What is the effect of not having that part?

7. **Memory Virtualization.** Mr Wayne doesn't remember how that whole multi-level paging thing works, so you're going to show him. Assume a 15-bit virtual address, 32-byte page size, and a two-level page table. The format of the page directory entry (PDE) and page table entry (PTE) is the same: a valid bit followed by a 7-bit page frame number (PFN). The page directory base register (PDBR) is set to decimal 127. Here is a partial dump of memory:

```
page   7: 08 0f 0d 05 05 0a 05 1a 1b 1b 05 10 09 04 16 0e 04 18 05 0d 03 10 0c 0e 0a
    0c 1d 09 0d 01 08 1b
page  19: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f c7 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
    7f 7f 91 7f f5 7f 7f
page  78: 7f 7f d3 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
    7f 7f 7f 7f 7f 7f 7f
page 106: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f c5 7f 7f 7f 7f 7f 7f 7f 7f 7f
    7f 7f 7f 87 7f 7f 7f
page 127: ab e9 a1 7f d6 81 f6 9a ea a9 bd ce b9 fb ad bb ba 7f d5 ac a4 9f 94 9e e1
    d8 7f e3 93 7f c6 ef
```

The first virtual address to translate is 0x238e. What happens when you try to load this virtual address? (If valid, what value do you get back?)

The second virtual address to translate is 0x2f30. What happens when you try to load this virtual address? (If valid, what value do you get back?)

The last virtual address to translate is 0x772d. What happens when you try to load this virtual address? (If valid, what value do you get back?)

8. **Memory Virtualization.** When running 50 jobs simultaneously, BatOS is going to need a lot more physical memory than Mr Wayne's computer has, so he plans to just use a portion of the hard disk's space (that he calls the Bat Cave) and not use physical memory at all. What effect would this have on his system's performance?

   You suggest that he instead add a reference bit to each page table entry and keep a few active pages in physical memory and the less-recently-used ones in the Bat Cave. Explain how to use a reference bit to figure out which pages have been used recently.

9. **Memory Virtualization.** BatOS's library function `bat_alloc()` behaves like the `Mem_Alloc()` function you have written, but Mr Wayne has decided not to implement coalescing to save some processor time. "It doesn't work!" says Mr Wayne. "In the middle of my test program I tried to allocate some space and it told me there wasn't room, but all the memory was free!" What must that program have done?

   Mr Wayne agrees that not ever coalescing memory is a bad idea, and proposes having coalescing triggered by a `bat_alloc()` call that encounters the problem you just described. You disagree: show a situation using **Best Fit** policy where constant coalescing would succeed but BatOS's triggered coalescing fails.

10. **Concurrency.** Mr Wayne shows you his pseudocode implementation for lock operations in BatOS:

```
void lock(lock_t *lock) {
  while(first_time_slice) {
    if (xchg(lock->flag,1) == 0)
      return;
    else
      ; // spin
  }
  while (xchg(lock->flag,1) == 1)
    yield();
}
```

```
void unlock(lock_t *lock) {
  xchg(lock->flag,0);
}
```

How is this different from a typical spin lock? Does it work? How does it change the behavior of the lock?

11. **Concurrency.** Condition variables are queues that threads can wait in for certain conditions to be true, and can be useful in multi-threaded programs. Unlike the standard condition variables from the `pthread` library, BatOS's `wait()` call doesn't require the calling thread to pass it a lock. Explain to Mr Wayne why condition variables need access to a lock when they `wait()`.

12. **Concurrency.** As you may recall, Mesa semantics treat a `signal()` as a hint to that thread; a `signal`ed thread is moved from a "blocked" state to a "ready" state and can therefore be scheduled by the OS again. BatOS uses Hoare semantics (stop giggling), wherein a `bat_signal()` immediately interrupts itself and transfers control to the `signal`ed thread. Is there anything Mr Wayne *should* change about his multi-threaded programs? Is there anything he *can* change?

13. **Concurrency.** Mr Wayne's last question for now has to deal with passing information from one process to another via some sort of *bounded buffer*.

```
void *producer(void *arg) {
  while (1) {
    lock(&lock);              // p1
    while (numfull == max)    // p2
      wait(&cond, &lock);     // p3
    fill(buffer);             // p4
    bat_signal(&cond);        // p5
    unlock(&lock);            // p6
  }
}
```

```
void *consumer(void *arg) {
  while(1) {
    lock(&lock);              // c1
    while (numfull == 0)      // c2
      wait(&cond, &lock);     // c3
    int tmp = get(buffer);    // c4
    bat_signal(&cond);        // c5
    unlock(&lock);            // c6
    process(tmp);             // c7
  }
}
```

Explain to Mr Wayne why this solution won't work in general; include a trace through the code (like we did in class) to demonstrate.