Name:

CSCI 3753: Operating Systems
Spring 2014

<u>Midterm Exam</u>

03/11/2014

<u>Answer all questions in the space provided</u>

<u>Multiple Choice Questions:</u> Choose one option that answers the question best.
[30 Points]

1. Extended machine view of operating systems is

   - A. OS provides mechanisms to deal with the complexity of hardware
   - B. OS provides support for developing applications for a computing system
   - C. OS allows sharing and effective utilization of computing system resources
   - D. OS provides equivalent of a virtual machine that is easier to use
   - E. OS provides support for security and protection

2. What is the problem with the following implementation of mutual exclusion?

   Disable interrupts
   Critical section
   Enable interrupts

   - A. Two or more processes may execute in the critical section concurrently, if there are multiple CPUs.
   - B. A process in critical section may block every other process, even when the other processes are not trying to enter the critical section.
   - C. A process may omit Enable interrupts, causing the entire computing system to hang.
   - D. A, B and C.
   - E. B and C, but not A.

3. A thread pool

   - A. improves overall performance of an interactive application in the long run
   - B. improves average response time of an interactive application in the long run
   - C. may result in more threads in an application than needed
   - D. A, B and C
   - E. A and B, but not C

4. Which of the following is FALSE about a loadable kernel module (LKM)

   - A. It is an object file that contains code to extend a running kernel
   - B. It can be incorporated in a kernel without any need for a kernel re-compilation
   - C. As more and more LKMs are added, the OS kernel may become inefficient
   - D. Once added, it cannot be removed from a kernel without re-compilation
   - E. It can be used to add new device drivers as well as new system calls

5. A trap instruction

   - A. causes a hardware interrupt
   - B. changes the processor mode to supervisor mode
   - C. changes the processor mode back to user mode
   - D. A and B
   - E. A, B and C

6. Difference between user level threads and kernel level threads with in a process is

   - A. kernel levels threads share their stacks while user level threads do not
   - B. kernel levels threads share their heaps while user level threads do not
   - C. OS is aware of kernel level threads, but not user-level threads
   - D. OS schedule user level threads, but not kernel level threads
   - E. None of the above

7. Which of the following is NOT a part of process state?

   - A. Program counter value
   - B. List of open files
   - C. Number of processes it has forked
   - D. Its current priority
   - E. Its process ID

7. Which of the following statement is FALSE about O(N), O(1) and CFS schedulers in Linux?

   - A. O(1) scheduler has a constant time complexity while CFS has logarithmic time complexity
   - B. O(1) scheduler works better in practice than O(N) scheduler
   - C. O(N) scheduler is used in Linux versions prior to version 2.6
   - D. CFS uses red-black tree instead of runqueues
   - E. CFS works better in practice than O(1) scheduler

8. Direct I/O with interrupts

- A. relieves CPU from data transfer between memory and device registers
- B. relieves CPU from (busy) waiting for the I/O device to complete I/O
- C. requires an extra micro-controller
- D. enables device-independent I/O interface
- E. Both A and B

9. IPC using shared memory

- A. requires processes to agree on a key name in advance
- B. uses send and receive primitives
- C. requires one process to create a shared memory segment using shmget( ), while the other process to attach to the existing shared memory usinh shmat( )
- D. A and C but not B
- E. None of the above

10. Which of the following is FALSE about binary semaphores?

- A. It cannot be used for process synchronization
- B. It is as expressive as a monitor
- C. It is as expressive as a regular semaphore
- D. It can be used to solve the dining philosophers problem
- E. It can be used to solve the producer consumer problem

Short Answer Questions [25 Points]:

1. In Linux, explain the usage of schedule( ) and switch_to( ) kernel functions in implementing a context switch.

   The schedule( ) function is invoked after a timer interrupt, if a new process needs to be scheduled. This function updates ready_queue, run_queue etc, and then calls switch_to( ) function. The switch_to( ) saves the context of the running process its kernel stack and loads the state of the new process. As soon as this is done, the new process starts executing.

2. Explain two ways that I/O can be overlapped with CPU execution and how they are each an improvement over not overlapping I/O with the CPU.

   Interrupt driven I/O: CPU initiates the I/O and sets up an interrupt handler. After that it performs other useful work in parallel with I/O data transfer being performed by I/O device. When I/O data transfer is complete, the CPU is interrupted to complete the remaining work for completing the I/O.

   DMA based I/O: Similar to interrupt-driven I/O with the addition that DMA controller manages the data transfer between memory and device registers.

3. Describe the steps involved in a multi-stage procedure for bootstrapping an OS.

   Step 1: Power on self test: Check various hardware components
   Step 2: BIOS looks for a device to boot from: sequence may be prioritized
   Step 3: BIOS finds a hard disk drive to boot from: Looks for Master Boot Record (MBR) in the disk, which contains a primitive boot loader
   Step 4: Primitive boot loader then loads a secondary stage boot loader

4. Process P1 is to execute statements S1 and S2; process P2 is to execute statements S3 and S4. Process P1 must not terminate until P2 has executed statement S3. A colleague gives you the following program:

```
P1:                          P2:
  S1;                          S3;
  S2;                          wakeup(P1);
  sleep( );                    S4;
```

   Is the solution correct? Explain your answer.

   No, this solution is not correct. If P2 executes wakeup( ) before P1 executes sleep (), P1 will be blocked indefinitely.

5. What are I/O intensive and CPU intensive processes? How can this distinction be used in making scheduling decisions? How can an operating system determine if a process is I/O intensive or CPU intensive?

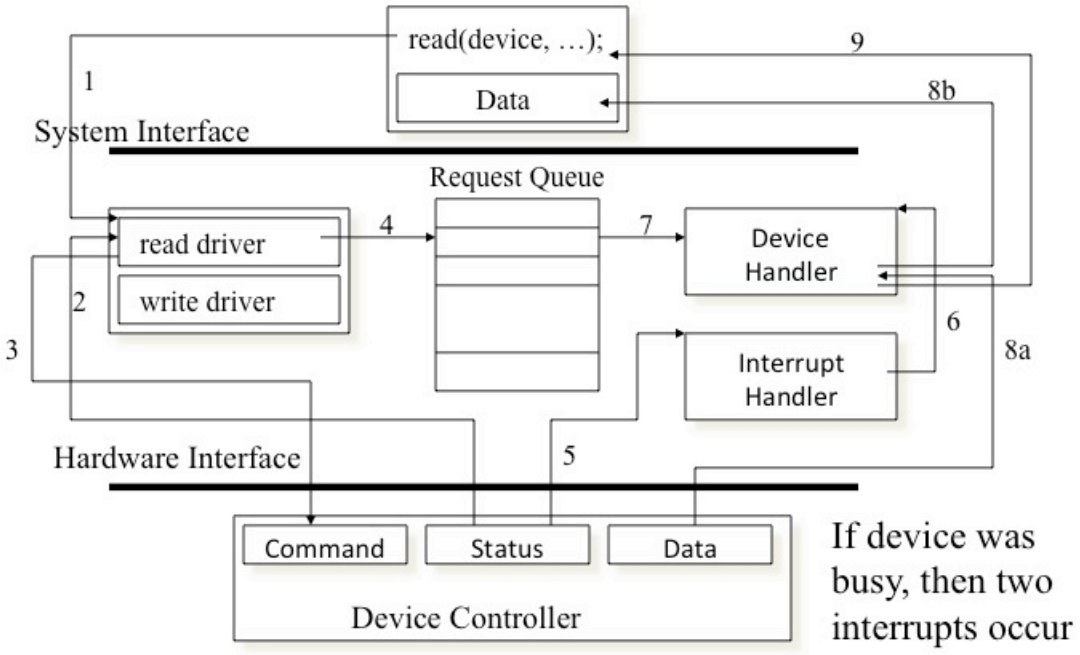
   An I/O intensive process spends most of its time doing I/O and very little time executing on CPU. A CPU intensive process spends most of its time executing on CPU and very little time doing I/O. Giving higher priority to I/O intensive processes enables higher parallelism between CPU and I/O devices, and thus improves overall performance. A process typically alternates between I/O intensive behavior and CPU intensive behavior during its execution. When a process is behaving as I/O intensive, it does not use all of the time slice allocated to it, while it uses its entire time slice when behaving as CPU intensive. Thus, OS monitors processes during their execution. If a process uses up its entire time slice, it is designated as CPU intensive at that moment and its priority is lowered, while if a process does not use all of its time slice, it is designated as I/O intensive at that moment and its priority is raised.

Problems

1. **[10 Points]** An *EventPair* object synchronizes a pair of threads (a *server* and a *client*) for a stream of request/response interactions. The server thread waits for a request by calling *Event-Pair::Wait()*. The client issues a request by placing data in shared memory and calling *EventPair::Handoff()*. *Handoff* wakes up the server thread and simultaneously blocks the client to wait for the reply. The server thread eventually places the reply in shared memory and calls *Handoff* again; this wakes up the client to accept the response, and simultaneously blocks the server to wait for the next request. Show how to implement *EventPair* using semaphores.

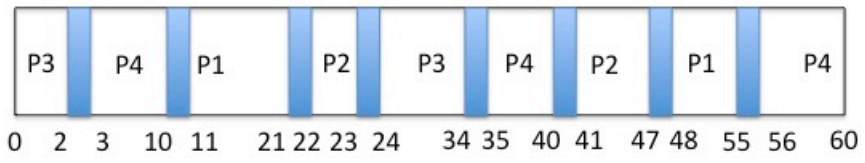
```
typedef struct {
   int toggle;
   Semaphore s1, s2, mutex;
} EventPair;

EventPair::Init( )
{
   toggle = 0; s1 = 0; s2 = 0; mutex = 1;
}

EventPair::Wait( )
{
   wait(s1);
}

EventPair::Handoff( )
{
   wait(mutex);
   if (toggle == 0) {
      toggle = 1;
      signal(s1);
      signal(mutex);
      wait(s2);
   }
   else{
      toggle = 0;
      signal(s2);
      signal(mutex);
      wait(s1);
   }
}
```

2. **[10 Points]** Draw and label a figure to show the sequence of steps in a *read()* operation from a disk, from the application first calling a *read()* through the OS processing the *read()* to the final return from the *read()* call upon completion of the disk operation. Assume interrupt-driven I/O. Your answer should include components such as the device controller, interrupt handler, device handler, device driver and any other OS components you deem appropriate to add.



3. **[10 Points]** Four non-realtime processes, P1, P2, P3 and P4 are running on a Linux system that is using O(1) scheduler. Assume that the time slice is 10 ms long, context switch time is 1 ms, and the priorities are updated by one place based on CPU or I/O bound nature of the process. Show the Gantt chart for the execution of these four processes and calculate the average turnaround time and average response time.

Process	Initial Priority	Execution Time (ms)	Additional Description
P1	113	17	P1 doesn't block during execution
P2	114	7	P2 performs I/O once for 4 ms after 1 ms of execution
P3	107	12	P3 performs I/0 once for 5 ms after 2 ms of execution
P4	108	16	P4 performs I/O twice: first for 3 ms after 7 ms of execution and then for 2 ms after another 5 ms of execution



Average turnaround time = (55 + 47 + 34 + 60)/4 = 49 ms

Average response time = (11 + 22 + 0 + 3)/4 = 9 ms

4. **[15 Points]** [Sleeping barber problem] A barber shop has one barber, one barber chair, and *n* chairs for waiting customers, if any, to sit on. If there are no customers present, the barber sits down in the barber chair and falls asleep. When a customer arrives, he has to wake up the sleeping barber. If additional customers arrive while the barber is cutting a customer's hair, they either sit down (if there are empty chairs) or leave the shop (if all chairs are full). When the barber shows up for work in the morning, he/she executes the function *barber* that enables him/her to cut customers' hair throughout the day. A customer executes the function *customers* to get a haircut. Your task is to write these two functions: *barber*( ) and *customer*( ). Use monitors for synchronization and to prevent race conditions.

```
void barber( )
{
   while (1) {
     sleeping_barber.wait_for_customer( );
          cut hair
   }
}
```

```
void customer( )
{
   int haircut;
   sleeping_barber.get_haircut(haircut);
   if (haircut == 0) get haircut;
   if (haircut == 1) shop is full;
   if (haircut == 2) barber hasn't arrived yet;
}
```

```
Monitor sleeping_barber
{
   const int N = 50; //Number of chairs
   int customer_waiting;
   boolean barber_sleeping, barber_avail;
   condition customer_cond, barber_cond;

   wait_for_customer( )
   {
      barber_avail = True;
      if (cutomer_waiting == 0) {
         barber_sleeping = True;
         barber_cond.wait( );
         barber_sleeping = False;
      }
      else{
         customer_cond.signal( );
      }
   }

   void get_haircut(int &full)
   {
      if (barber_avail == False) {
        full = 2;
        return;
      }
      if (barber_sleeping) {
        full = 0;
        barber_cond.signal( );
      }
      else if (customer_waiting < N) {
         customer_waiting++;
         customer_cond.wait( );
         customer_waiting--;
         full = 0;
      }
      else
         full = 1;
   }

   void init_monitor( )
   {
      customer_waiting = 0;
      barber_sleeping = True;
      barber_avail = False;
    }
}
```