Name:

CSCI 3753: Operating Systems
Fall 2016
Midterm Exam Solutions

Answer all questions in the space provided

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

1. Resource manager 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. All of the above

2. Difference between multiprogramming and multitasking is
   - **A. In multiprogramming, a long running CPU-bound process may significantly delay the execution of other processes, while in multitasking, this cannot happen.**
   - B. In multitasking, a process can be preempted only when it blocks for I/O, while in multiprogramming, a process can be preempted any time.
   - C. Time needed to do a context switch is much larger in multiprogramming than in multitasking.
   - D. There is no difference between multiprogramming and multitasking if none of the processes block during execution.
   - E. A and D only.

3. Which of the following is FALSE about trap table?

   - A. A trap table is used to implement systems calls.
   - B. A trap table resides in the OS kernel.
   - **C. A user can modify trap table entries by writing an LKM.**
   - D. Entries in a trap table can be accessed only in supervisor mode.
   - E. A user process can access trap table entries using system calls.

4. The processor mode bit in modern computing systems

    - A. allows OS to implement interprocess communication using shared memory.
    - B. allows OS to implement multi-threading inside the kernel.
    - **C. allows OS to protect the kernel from interference from the user program.**
    - D. ensures that user programs cannot execute kernel functions.
    - E. None of the above.

5. The /proc directory

    - A. allows applications and users to peer into the kernel's view of the system.
    - B. Contains a hierarchy of special files which represent the current state of the kernel.
    - C. provides a mechanism for the kernel and kernel modules to send info to processes in user space.
    - D. contains a special type of files called a virtual file.
    - **E. All of the above.**

6. Which of the following is NOT included in process control block?

    - A. program counter value
    - B. list of open files
    - C. priority
    - **D. cached data**
    - E. stack

7. Advantages of threads over processes include

    - **A. lower context switch time**
    - B. no possibility of race conditions
    - C. sharing of heap and stack
    - D. smaller code size
    - E. All of the above

8. Which of the following is FALSE about IPC via pipes?

    - A. Basic primitives are send( ) and receive( ).
    - B. Communication can be blocking or non-blocking.
    - C. Pipes can be anonymous or named.
    - **D. Pipes can be used for only one-way communication.**
    - E. IPC via pipes is slower than IPC using shared memory.

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 using shmat( )
- D. A and C but not B
- E. None of the above

10. Which of the following is FALSE about Pthread mutex?

- A. It cannot be used for process synchronization
- B. It can be used for implementing mutual exclusion
- C. It is not as expressive as a binary semaphore
- D. Deadlocks can occur if it is not properly used
- **E. It can be used to solve the producer consumer problem**

<u>Short Answer Questions</u> [30 Points]:

1. The following code is executed by 10 different processes that all share the integer variable *counter* whose initial value is 5. Explain through an example how there is a possibility of race condition in this code.

   counter++;

<u>Assembly code:</u>
(1) reg1 = counter
(2) reg1 = reg1 + 1
(3) counter = reg1

Process 1 executes (1) and (2) and is preempted
Process 2 executes (1) (2) and (3) and exits
Process 1 is re-scheduled and executes (3)

counter is increased by 1 while two processes have incremented it
2 Points for correctly getting the assembly code
2 Points for getting the sequence of steps correct
1 point for identifying the discrepancy

2. Using a simple system call as an example (e.g. getpid, or uptime), describe what is generally involved in providing the result, from the point of calling the function in the C library to the point where that function returns.

(1) The library function invokes the appropriate system call, e.g. getpid
(2) The system call invokes the trap instruction that changes the processor mode bit to zero and indexes into the trap table to invoke the getpid function in the kernel; The getpid function is executed
(4) Processor mode bit is changed to 1 before the control is returned to the user program

1 Point for (1) – difference between library function and system call
2 Points for (2) – processor mode bit change and indexing in the trap table
2 Points for (3) – processor bit change and return to user program

3. 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.

   2 Points for identifying the two ways
   3 Points for proper description

4. Provide a step-by-step description for adding a new device in Linux operating system without requiring recompiling the kernel.

   (1) Write the device driver, compile it to create an LKM (.ko file)
   This LKM contains an initialization routine init_module( ) function that registers various device functions contained in the LKM with the kernel using appropriate kernel functions such as register_chrdev, register_blkdev, etc.
   This registration fills the entry table in the kernel (dev_func_i[j]) with appropriate function pointers

   (2) Call insmod command that gets a unique device number for the new device and invokes the init_module systems call to load the LKM. This system call invokes the LKM's initialization routine.

   1 Point for identifying LKM
   2 Points for details of LKM as described in (1)
   1 Point for identifying insmod
   1 Point for details as described in (2)

5. Both CPU and DMA controller move data to/from main memory. Describe three ways they can share access to memory.

- Burst mode: While DMA is transferring, CPU is blocked from accessing memory
- Interleaved mode or "cycle stealing": DMA transfers one word to/from memory, then CPU accesses memory, then DMA, then CPU, etc…
- Transparent mode: DMA only transfers when CPU is not using the system bus

3 Points for correctly identifying the three ways
2 Points for correct description

6. In round robin scheduling, discuss the tradeoffs in choosing a large time slice vs a small time slice.

A large time slice reduces the number of context switches and hence reduces the overall overhead due to context switch times resulting in higher CPU utilization. On the other hand, a large process can occupy the CPU for a relatively larger amount of time whenever it gets the CPU, so the average wait time and average response time of the processes is increased.
So, a large time slice results in higher CPU utilization but larger wait and response times, while a small time slice results in poorer CPU utilization but smaller waut and response times.

2 Points for identifying CPU utilization
2 Points for identifying wait and response times
1 Point for correctly associating advantages/disadvantages with large/small time slices

Problems

1. **[8 Points]** Suppose a computer center has two printers, A and B, that are similar but not identical. There are three kinds of processes, K1, K2 and K3 that use these printers with the following rules: K1 processes can only use printer A, K2 processes can only use printer B, and K3 processes can use either A or B.

   A process calls *request* function to request a printer. This function returns the identity of a free printer. After using that printer, the process returns it by calling the *release* function. Implement *request* and *release* functions for the three kinds of processes. Use semaphores for synchronization. Your solution should be fair and deadlock free assuming that a process using a printer eventually releases it.

```
Semaphore A_sem = 1, B_sem = 1;

Printer request_K1( )
{
   wait (A_sem);
   return A;
}
void release_K1( )
{
   signal(A_sem);
}

Printer request_K2( )
{
   wait (B_sem);
}
void release_K2( )
{
   signal(B_sem);
}

Printer request_K3( )
{
   wait (A_sem);
   wait (B_sem);
}
void release_K3( )
{
   signal(A_sem);
   signal(B_sem);
}
```

2 Points for declaring and initializing the semaphores
2 Points each for request/release functions for K1, K2 and K3

2. **[12 Points]** You are the organizer of a gaming exhibit where you want attendees to play your startup's new game demo. You model the attendees as threads, called players, and your job is to synchronize access to a single copy of the game, as follows:

- When a player arrives, he or she waits in a waiting area.
- When there are four or more players waiting to play, you allow exactly four of them to exit the waiting area to start playing. The four players leave the waiting area and approach the game console.
- When a player reaches the game console, the player waits until all four players are at the console, at which point all four players begin playing.
- Players may finish playing at any time. However, you cannot allow any new player to approach the game console until all four players have left.

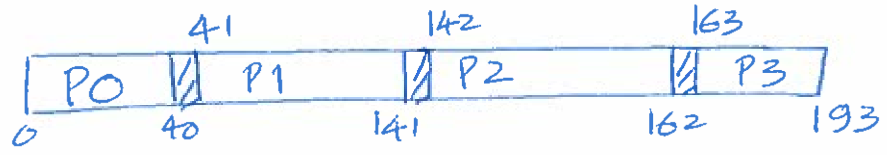
Your task is to write a function *play*( ) that a player calls when he/she wants to play. Use monitors for synchronization.

```
void play( )
{
   myGame.start_playing( );
        play the game
   myGame.end_playing( );
}

monitor myGame
{
   int count_w; //# of players waiting in the wait area
   int count_c; //# of players waiting at the console;
   int count_p; //# of players playing;
   condition c_area; // condition to block players in the waiting area
   condition c_console; /condition to block players at the console

   void start_playing( )
   {
      count_w++;
      if (count_w < 4 || count_p > 0 || count_c > 0) c_area.wait( );
      count_c++; count_w--;
      if (count_c < 4) c_area.signal( );
      count_p++;
      if (count_p < 4) c_console.wait( );
      count_c--;
      if (count_c > 0) c_console.signal( );
   }
void end_playing( )
   {
      count_p--;
      if (count_p == 0 && count_w > 3) c_area.signal( );
   }

   void init_code( )
   {
      count_w = count_p = count_c = 0;
   }
}
```

2 Points for getting the play ( ) function right. <play the game> must be here sandwiched between start_playing and end_playing functions
1 Point for getting the structure of the monitor right: keyword monitor, variable declarations, functions
1 Point for correctly initializing the monitor variable (init_code)
3 Points for declaring three counters and two condition variables
2 Point for making sure that exactly four players move to the console
2 Point for making sure that players start playing when there are exactly four players at the console
1 Point for making sure that the last player to finish playing signals a waiting player in the waiting area if needed

3. **[20 Points]** Four processes, P0, P1, P2 and P3 are created in a system at times 0, 10, 50 and 60 respectively. Their CPU time requirements are 40, 100, 20 and 30 time units respectively. Assume that the context switching time is 1 time unit.

(a) Show the Gantt chart for the execution of these four processes, if the FCFS scheduling algorithm is used. Calculate the average turnaround time.

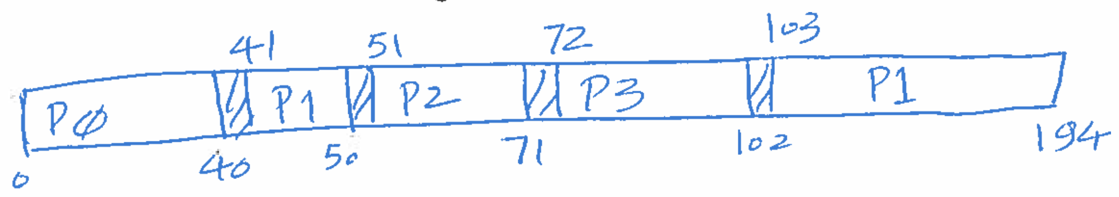


avg turn around time = $\frac{40 + 131 + 112 + 133}{4}$

3 Points for getting the chart right

2 Points for avg. turn around time

(b) Show the Gantt chart for the execution of these four processes, if the (preemptive) shortest job next (shortest remaining time first) scheduling algorithm is used. Calculate the average wait time.



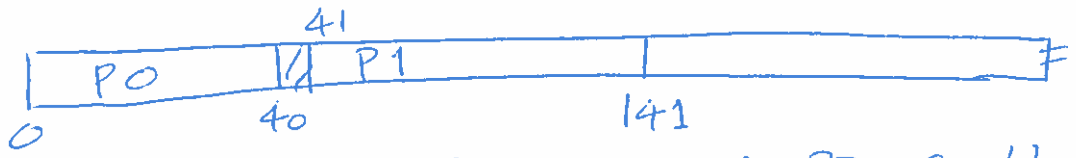
avg. wait time = $\frac{\emptyset + (31 + 53) + 1 + 12}{4}$

2 Points for the chart

1 Point for ~~pe~~ pre-empting P1 at 5Ø

2 Points for average wait time

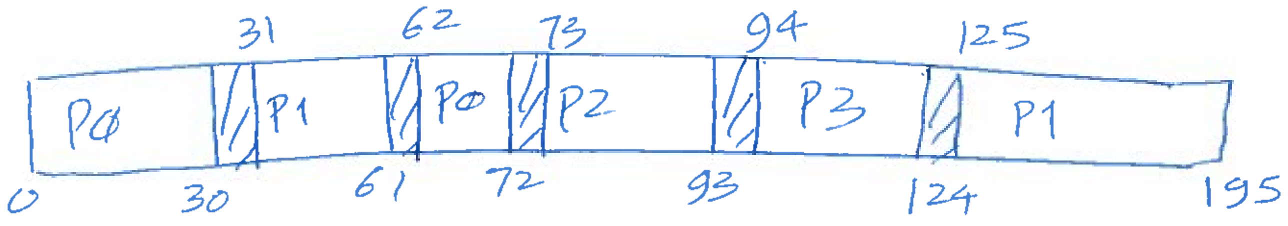
(c) Show the Gantt chart for the execution of these four processes, if the (non-preemptive) earliest deadline first scheduling algorithm is used. The completion deadline for these processes are 100, 310, 40 and 60 respectively. Do all processes meet their deadlines?



Deadlines of P2 and P3 can't be met, so they are not admitted.

2 Points for the chart
2 Points for identifying P2 and P3
1 Point for mentioning that P2 and P3 are not admitted

(d) Show the Gantt chart for the execution of these four processes, if a pure round-robin scheduling algorithm is used with time slice of 30 time units. Calculate the average response time.



avg. response time

$$= \frac{0+21+23+34}{4}$$

2 Points for the chart
1 Point for scheduling P∅ at 62
2 Points for avg. response time