Matriculation Number: | | | | | | | | | | CS2106

NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
MIDTERM EXAMINATION FOR
Semester 1 AY2011/2012

CS2106 Introduction to Operating Systems

October 2011 — Time Allowed 1.5 hours

The tutorial group I regularly attend is: (please circle one)

1. Mon 0900
2. Mon 1000
3. Mon 1100
4. Mon 1400
5. Mon 1500
6. Mon 1600
7. Thu 1000
8. Thu 1100

INSTRUCTIONS TO CANDIDATES

1. This exam paper contains 13 questions and comprises 10 printed pages, including this page.
2. The total marks for this examination is 80. Answer **ALL** questions.
3. Write **ALL** your answers in the box provided. Please indicate clearly (with an arrow) if you use any space outside the box for your answer.
4. Write succintly and clearly. If you make any additional assumption, please state your assumption clearly.
5. This is an **CLOSE BOOK** examination, but you are allowed to bring in **one sheet of double-sided A4 size paper** with notes.
6. Write your matriculation number in the boxes above and on the top-left corner of every page.

EXAMINER'S USE ONLY		
Q1-10	40	
Q11	10	
Q12	15	
Q13	15	
TOTAL	80	

Part I

Multiple Choice Questions (40 points)

For each of the question below, select the most appropriate answer and **write your answer in the answer box**. Each question is worth 4 points.

It is possible that none of the answers provided are appropriate. If you believe that NONE of the answers are appropriate, put an X in the answer box.

It is also possible that multiple answers are equally appropriate. In this case, pick ONE and write the chosen answer in the answer box. Do NOT write more than one answers in the answer box.

1. One decision that OS designers need to make is whether to implement a particular OS component inside the user space or inside the kernel space. Which of the following is NOT a factor to consider when making this decision?

   A. The frequency of interaction between the component and the user applications

   B. The frequency of interaction between the component and the rest of the kernel

   C. The potential damage that a bug in the component can cause

   D. The ease of use of the interface provided by the component

   Answer: [ ]

2. Consider the following program.

```
int main()
{
        int x = 1;
        fork();

        x = x + 1;
        fork();

        printf("%d ", x);
}
```

   What will be printed by the program above when it is executed?

   A. 2 2 2 2

   B. 1 2 3 4

   C. 2

   D. 4

   Answer: [ ]

3. Which of the following statement is FALSE?

   A. A process can create another process with fork()

   B. A process can terminate another process with kill()

   C. A process can block another process with wait()

   D. A process can terminate itself with exit()

Answer: ☐

4. Consider two processes P and C in a Linux-based operating system, where P is the parent of C.

   Which of the following statement is FALSE?

   A. If C calls `exit()` before P calls `wait()`, C becomes a zombie process.

   B. If P calls `exit()` before C calls `exit()`, C becomes an orphan process.

   C. If C calls `exec()`, P is no longer the parent process of C.

   D. If C calls `exit()`, a SIGTERM signal is sent to P.

Answer: ☐

5. You are implementing an application that needs to spawn off a number of tasks. You need to make a decision to implement these tasks either as processes or as threads.

   Which of the following statement is NOT a good guideline to making the decision?

   A. If you have a large number of tasks, then using multiple threads is more efficient.

   B. If the tasks share much common code, then using multiple threads is more efficient.

   C. If the tasks involves executing potentially buggy code, then using multiple processes is more robust.

   D. If much communication is needed among the tasks, then using multiple processes is more efficient.

Answer: ☐

6. Consider the following different possible pseudocode taken from two processes, where A, B are blocks of code, and S is a semaphore initialized to 0.

   Which of the following sequence ensures that A always runs BEFORE B?

   A.
   ```
   Process 1:     Process 2:
     A;             up(S);
     down(S);       B;
   ```
   B.
   ```
   Process 1:     Process 2:
     A;             down(S);
     up(S);         B;
   ```
   C.
   ```
   Process 1:     Process 2:
     down(S)        down(S)
     A;             B;
     up(S);         up(S);
   ```
   D.
   ```
   Process 1:     Process 2:
     up(S)          down(S)
     A;             B;
   ```

   Answer: ☐

7. The following pseudocode shows a solution to the Dining Philosopher problem with N philosophers, using an array of N semaphores (each initialized to 1) to represent N chopsticks (for $N > 2$).

   ```
   while (1)
      think;                       // think
      down(chopstick[i]);          // pick up left chopstick
      down(chopstick[i+1 \% N]); // pick up right chopstick
      eat;                         // eat
      up(chopstick[i]);            // put down left chopstick
      up(chopstick[i+1 \% N]);   // put down right chopstick
   ```

   Which of the following statements is TRUE?

   A. The solution is deadlock-free.

   B. The solution is starvation-free.

   C. The solution ensures fairness among philosophers.

   D. The solution ensures that no two philosophers eat at the same time.

   Answer: ☐

8. Two computer systems, A and B, use the same round-robin scheduler and are exactly identical (including the scheduler parameter and the set of processes running on the systems), except that A has a faster I/O subsystem.

   Which of the following statement is FALSE?

   A. CPU utilization on A will be higher than B.

   B. Throughput of A will be higher than B.

   C. Average turnaround time of processes in A will be higher than B.

   D. Average time a process spend waiting in queue will be lower in A than B.

   Answer: ☐

9. Which of the following statement is FALSE?

   A. Round robin (RR) scheduler behaves like first-come first-serve (FCFS) scheduler for infinitely large time quantum.

   B. Multi-level feedback queue (MLFQ) scheduler behaves like round robin (RR) scheduler if there is only one priority level.

   C. Shortest remaining time first (SRTF) scheduler behaves like shortest job first (SJF) algorithm if the set of processes are fixed (no new process is created) and the processes are completely CPU bound.

   D. Multi-level feedback queue (MLFQ) scheduler behaves like round robin (RR) scheduler if the set of processes are fixed (i.e., no new process is created) and the processes are completely CPU bound.

   Answer: ☐

10. There are two files A and B in your Linux home directory with identical content. After you updated the content of file A; you noticed that the content of file B is updated as well (with the same content of A). You deleted file A, but file B is still accessible.

    What is the relationship between A and B?

    A. A is a copy of B, created with `cp` command.

    B. A is renamed from B, created with `mv` command.

    C. A is hard link of B, created with `ln` command.

    D. A is soft link of B, created with `ln -s` command.

    Answer: ☐

Part II

Short Questions (40 points)

Answer all questions in the space provided. Be succinct and write neatly.

11. (10 points) To ensure mutual exclusion of a critical region between two processes, Process 0 and Process 1, the following implementation of `enter()` and `leave()` are called before entering and leaving a critical region respectively. Before entering a critical region, Process 0 calls `enter(0, 1)`, while Process 1 calls `enter(1, 0)`. When leaving the critical region, Process 0 calls `leave(0)`, while Process 1 calls `leave(1)`. Both `interest[0]` and `interest[1]` are set to 0 initially.

```
void enter(int self, int other) {
    turn = other;
    interest[self] = 1;
    while (interest[other] == 1 && turn == other);
}

void leave(int self) {
    interest[self] = 0;
}
```

The difference between this implementation and the original Peterson's Algorithm is that the code sets the `turn` variable first, before setting the `interest` variable.

Does the implementation above properly ensures mutual exclusion, i.e., no two processes will enter the critical region at the same time?

Either argue why the mutual exclusion property is ensured, or give an example sequence of execution that leads to a violation of that property.

12. (15 points) You are asked to implement two threads that share a bounded buffer. The first thread, the producer, repeatedly produces one item to be stored in the buffer (by calling produce()). The second thread repeatedly consumes one item from the buffer (by calling consume()).

You recall that this scenario is exactly the producer-consumer problem you learnt in CS2106. You look up your notes, and found the following solution to the problem.

```
semaphore free_slots = N
semaphore used_slots = 0
semaphore access_buffer = 1

Producer:                       Consumer:
    while (true)                   while (true)
        down(free_slots)               down(used_slots)
        down(access_buffer)            down(access_buffer)
        produce()                      consume()
        up(access_buffer)              up(access_buffer)
        up(used_slots)                 up(free_slots)
```

Happily, you proceed to implement the pseudocode above, but found that, alas, the platform you are using does not support semaphore! You only have access to a thread library that provides the following operations for mutexes and condition variables, with the same semantics as the mutex and condition variable API provided by the POSIX thread library.

```
mutex m   // declare and initialize a mutex (initially unlocked)
lock(m)   // lock the mutex m
unlock(m) // unlock the mutex m

cond c      // declare and initialize a condition variable c
wait(c, m) // wait on condition variable c, unlocking m if m is locked
            // while waiting; relocking m upon woken up.
signal(c)  // wakes up a thread waiting on condition variable c
```

Rewrote the pseudocode for the producer-consumer threads above with mutex and condition variables, using only the operations provided above. You need not worry about creating and joining the threads. You may assume operations to check whether the buffer is full or is empty is available.

13. (15 points) (a) (2 points) Give an argument that favours the use of a large time quantum in a pre-emptive scheduler.

. . . .

. . . .

. . . .

. . . .

(b) (2 points) Give an argument that favours the use of a small time quantum in a pre-emptive scheduler.

. . . .

. . . .

. . . .

. . . .

(c) (2 points) Explains how MLFQ attempts to achieve low turnaround time by approximating shortest job first algorithm.

. . . .

. . . .

. . . .

. . . .

An operating system uses multi-level feedback queue (MLFQ) with two priority levels, high and low, to schedule processes. Let's call the algorithm the *two-level feedback queue* algorithm, or 2LFQ for short.

2LFQ maintains two round-robin queues, one for each priority, and assigns the same time quantum T to every process, regardless of the priority level. 2LFQ works as follows. A new process enters the high priority queue. When a process at the high priority queue uses up its time quantum, it is preempted and joins the low priority queue. When a process at the low priority queue blocks for I/O, it joins the high priority queue. Otherwise, the process stays at the same priority level.

2LFQ always run the process at the head of the queue at the high priority queue, unless the queue is empty. In which case, the head of the queue at the low priority queue is chosen to run.

(d) (3 points) Suppose we use a smaller T for all processes in 2LFQ, would this result in less or more processes (compare to a larger T) in the high priority queue?

Justify your answer.

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

(e) (3 points) MLFQ aims to give interactive processes higher priority than batch (non-interactive) processes. Does 2LFQ achieve this objective?

Justify your answer. You may want to discuss your answer in relation to T.

. . . .
. . . .
. . . .
. . . .
. . . .
. . . .

(f) (3 points) Is 2LFQ a good approximation to the shortest job first algorithm? Justify your answer.

. . . .
. . . .
. . . .
. . . .
. . . .
. . . .

END OF PAPER

Stay Hungry. Stay Foolish.
— Steve Jobs (1955–2011)