COMP 252 Principles of Systems Software
Lecture Section 1

Fall Semester 1998

Midterm Examination

Date: October 27, 1998 Time: 09:00 - 10:15

Name: Solution Student ID: ______________ Email: __________

Instructions:

1. This examination paper consists of 7 pages and 6 questions.
2. Please write your name, student ID and Email on this page.
3. For each subsequent page, please write your student ID at the top of the page in the space provided.
4. Please answer all the questions within the space provided on the examination paper. You may use the back of the pages for your rough work.
5. Please read *each question very carefully* and answer the question clearly and to the point. Make sure that your answers are neatly written, readable and legible.
6. Show all the steps you use in deriving your answer, wherever appropriate.
7. For each of the questions assume that the concepts are known to the graders. Concentrate on answering to the point what is asked. Do not define or describe the concepts.

Question	Points	Score
1	10	
2	10	
3	10	
4	20	
5	25	
6	25	
TOTAL	100	

1. We know that I/O devices are very slow compared to the speed of the CPU. List two techniques that an operating system uses to keep the CPU busy while I/O operations are in progress. For each technique that you list, give a brief one or two sentence description of the technique. (10 points)

   Two techniques that can be used are:

   a) **Spooling:** In this case the I/O operations of one process are overlapped with the execution of another process. The CPU uses the disk as a buffer in performing I/O

   b) **Multiprogramming:** In this case, multiple programs are simultaneously resident in main memory. When one process issues an I/O request and gets blocked, the CPU is switched to another active process resident in memory. Time-sharing is an extension of multiprogramming where the CPU is frequently switched between processes

2. Recall the second solution to the too much milk problem. The corresponding solution for a system with two threads *A* and *B* is given below. Give a sequence of steps to show why this solution is incorrect (An example of a *step* is: Thread *A* leaves note *A*). (10 points)

   Thread A:

   ```
   leave Note A;
   while (Note B)
       no-op;
   if (no Milk)
       buy Milk;
   remove Note A;
   ```

   Thread B:

   ```
   leave Note B;
   while (Note A)
       no-op;
   if (no Milk)
       buy Milk;
   remove Note B;
   ```

   Consider the following sequence of events:

   a) Thread A leaves Note A

   b) Thread A gets interrupted and the CPU is switched to Thread B

   c) Thread B leaves Note B

   d) Thread B gets interrupted and the CPU is switched to Thread A

   e) Thread A executes the while loop, gets trapped into busy-waiting mode

   f) Thread A gets interrupted and CPU is switched to Thread B

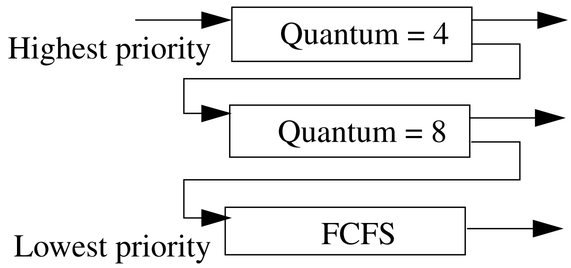
   g) Thread B executes the while loop, gets trapped into busy-waiting mode

   Neither Thread A nor Thread B can proceed forward because they are both trapped in their while loops waiting for the other to remove the note. This solution does not satisfy the progress requirement.

3. When the currently running process issues the system call '*sleep(1)*' to block itself for 1 second, indicate which of the following events will occur and their order of occurrence (i.e., which event occurs first, which event occurs second, etc.) before another user process resumes execution. Please fill in the boxes to the left with your answer. Please indicate the sequence using numbers 1, 2, 3, ... for those events that will occur. For those events that will not occur, please put a cross in the box. (10 points).

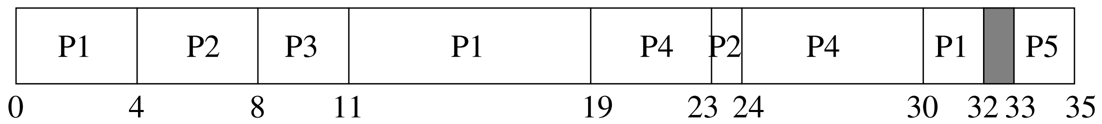
Answer	Event
X	a) The current process saves its context to its PCB (WILL NEVER OCCUR).
4	b) The kernel calls the sleep routine.
5	c) The kernel calls the scheduler.
6	d) The kernel checks the wait queue and puts any awakened processes back into the ready queue.
2	e) The kernel jumps to and executes the interrupt handler.
X	f) The kernel makes the current process a zombie process.
X	g) The kernel puts the current process into the ready queue.
X	h) The kernel puts the next process that resumes execution into the ready queue (WILL NEVER OCCUR).
8	i) The kernel restores the context of the next process that resumes execution from its PCB.
X	j) The kernel restores its last saved context (ONLY MEANINGFUL FOR KERNEL THREADS).
9	k) The kernel returns from interrupt.
3	l) The kernel saves the context of current process to its PCB.
X	m) The kernel saves its current context (ONLY MEANINGFUL FOR KERNEL THREADS).
7	n) The kernel selects the next process to resume execution from the ready queue.
X	o) The next process that resumes execution restores its context from its PCB (WILL NEVER OCCUR).
X	p) Software exception interrupt occurs.
1	q) Trap (i.e. software interrupt) occurs.
X	r) Timer interrupt occurs.

4. Consider a 3-level feedback queue with Round-Robin Scheduling used for the two high-priority queues and FCFS used for the lowest priority queue, as shown in the figure below. The scheduling is nonpreemptive:



Process	P1	P2	P3	P4	P5
CPU-burst	14	5	3	10	2
Arrival Time	0	2	4	15	33

a) Construct a Gantt chart depicting the process scheduling for the set of processes specified in the above table of processes. (15 points)



b) Calculate the average waiting time for the schedule constructed in Part a). (5 points)

P1 = (0 - 0) + (11 - 4) + (30 - 19) = 18
P2 = (4 - 2) + (23 - 8) = 17
P3 = (8 - 4) = 4
P4 = (19 - 15) + (24 - 23) = 5
P5 = (33 - 33) = 0

Average = 8.8

5. Giordano has opened a new outlet in Metro City Plaza in Tseung Kwan O. The operations at this store can be described as follows. Sales representatives wait until a customer is ready to buy. Once a customer is ready, the sales representative takes the merchandise to the cash register to process the sale. The store has a single cash register which the sales representative must access exclusively to process a purchase transaction. The customer waits until the sale is completed, before leaving. Once the sale is completed, the sales representative goes back to wait for another customer. This operation can be described using multiple threads representing sales representatives and customers. Implement a proper coordination mechanism between the customers and sales representatives using semaphores. For your convenience, a skeleton structure for the threads are given below:(25 points)

```
// Global variables and their initial values

Semaphore mutex; (Initial value = 1)
Semaphore CustomerArrive; (Initial value = 0)
Semaphore SaleCompleted; (Initial value = 0)


Sales() {

    while (TRUE) {

    // Wait until customer ready

    CustomerArrive->P();


    // Access cash register to
    // process sale

    mutex->P();
    Access Cash Register
    mutex->V();


    // Complete customer sale

    SaleCompleted->V();

    } // end While()

}
```

```
Customer() {

    // Indicate customer ready

    CustomerArrive->V();

    // Wait until sale completed

    SaleCompleted->P();

}
```

6. Consider the following two programs: (25 points)

Program P:

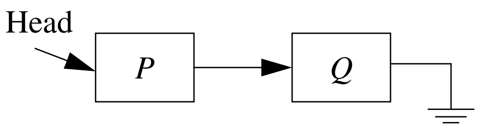
```
int x, a, b;

main() {
    x = 0;

    // kernel threads
    create kernel thread A;
    create kernel thread B;

    wait until threads A
    and B terminate;

    print a, b;
}

thread A() {
    x = x + 1;  // statement (1)
    yield();    // kernel thread yield
    a = x + 1;  // statement (2)
}

thread B() {
    x = x + 2;  // statement (3)
    b = x + 2;  // statement (4)
}
```

Program Q:

```
int x, c, d;

main() {
    x = 0;

    // PURE user threads
    create user thread C;
    create user thread D;

    wait until threads C
    and D terminate;

    print c, d;
}

thread C() {
    x = x + 3;  // statement (5)
    yield();    // user thread yield
    c = x + 3;  // statement (6)
}

thread D() {
    x = x + 4;  // statement (7)
    d = x + 4;  // statement (8)
}
```

Note that Program P creates **kernel threads** *and Program Q creates* **pure user threads**. *Also, when a thread is created, it does not run immediately. It is first put into the corresponding ready queue.*

Two processes are forked to execute the programs *P* and *Q* respectively (*P* is forked first). Before any of the processes start their execution, the *kernel ready queue* looks like this:



Both the kernel and the user thread library use FCFS scheduling policy.

Suppose during the program execution there are only two timer interrupts that force the kernel to do process/thread context switches. The two timer interrupts occur precisely between statements (3) and (4) and between statements (7) and (8) respectively. Besides these, there are no other external interrupts nor exceptions.

a) Show the actual execution order of the statements (i.e. Statement (1), (2), etc.) and the changes in the kernel ready queue.

Order	Kernel Ready Queue	Statement Being Executed	Process/Thread in CPU
1	(head) P - Q - ∇		
2	(head) Q - A - B - ∇		P
3	(head) A - B - ∇	(5)	Q
4	(head) A - B - ∇	(7)	Q
5	(head) B - Q - ∇	(1)	A
6	(head) Q - A - ∇	(3)	B
7	(head) A - B - ∇	(8)	Q
8	(head) A - B - ∇	(6)	Q
9	(head) B - ∇	(2)	A
10	(head) ∇	(4)	B

b) What are the final values of the variables a, b, c and d printed at the end the programs?

a = 4

b = 5

c = 10

d = 11