Name: **Student Id:**

CMPSCI 377: Operating Systems
Homework 3 (Mock Exam): Processes, CPU Scheduling, Synchronization

General instructions:

- Do not forget to put down your name and student number on the exam books.
- The exam is closed book and closed notes.
- Explain your answers clearly and be concise. Do not write long essays.
- You have 90 minutes to complete the exam. Be a smart test taker, if you get stuck on one problem go on to the next. Don't waste your time giving details that the question does not request.
- Show your work. Partial credit is possible, but only if you show intermediate steps.
- Good luck.

---

1. **Short answer questions** (15 pts)

   (a) (4 pts) Why do microkernel operating systems have a performance disadvantage over those that employ monolithic kernels?

   (b) (6 pts) What is a system call? Draw a figure that shows the steps involved in executing a system call.

   (c) (5 pts) Explain the steps performed by the kernel during a context switch.

2. **Processes and Threads** (30 pts)

   (a) (10pts) Draw a state transition diagram showing how a process transitions from one state to another during the course of its execution. Using this figure, list the states that the following program will visit during its execution:

```
 producer()
{
     acquire(lock);
     if(full(buffer))
        condition_wait(cond_var, lock);
     buffer[next] = item;
     release(lock);
}
```

   Assume that *lock* is free, full(buffer) returns true, and that another consumer calls a *condition_signal* after the *condition_wait* has been invoked by this producer.

(b) (10 pts) Using the *fork* and *wait* system calls, write a program in which a parent process forks **two** child processes and each child forks a grandchild. The grandchildren print "I am a grandchild" and exit. Each child process waits for their respective grandchildren to finish before exiting, and the original process waits for its two child processes to exit. You may use pseudo code for your solution.

(c) (10 pts) Consider a user-level threads program that uses two threads. If both threads are CPU-bound, then how much speedup do you expect to see for your program if you move it from a uniprocessor to a dual-core machine? Next, assume that one user-level thread is blocked on a semaphore, while the other remains CPU-bound. Will the process see any speedup if you move it to from a uniprocessor to a dual-core machine in this scenario? Explain your answer in in brief.

3. **CPU Scheduling** (15 points)

Compute the completion times and waiting times for the following set of jobs. Assume a zero context switch overhead, a time slice of 1 second, and that all jobs are compute bound.

			Completion Time			**Wait Time**		
Job	length	arrival time	FCFS	RR	SRTF	FCFS	RR	SRTF
1	50	0						
2	40	0						
3	30	0						
4	20	5						
5	10	10						

4. **Synchronization and Deadlocks** (40 points)

(a) (6 pts) Consider the following implementation of a lock using the test&set instruction that we discussed in class:

```
Lock::Acquire()
{
  // if lock is busy, do nothing
  while(test&set(value) == 1);
}
```

Explain the purpose of the while loop in the Acquire() method. What kind of performance or efficiency problem does this implementation suffer from?

(b) (14 pts) Consider the following system state for four processes P_0, P_1, P_2 and P_3, and three resources A, B and C. Using Bankers algorithm, determine whether the following state is safe or unsafe. Explain your answer.

	Max			Allocation			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	0	1	0	0	0			
P_1	1	7	5	1	0	0			
P_2	2	3	5	1	3	5			
P_3	0	6	5	0	1	3			
total				2	4	7	1	5	3

(c) (20pts) *Too Little Milk:* You and your two room-mates decide to take turns to buy milk from the grocery store to solve the problem of not having enough milk. Each time you run out of milk, you must ensure that the three of you take turns in round-robin order: you, your room-mate 1, your room-mate 2, you, your roomate 1, etc. Write three routines using **semaphores** to take turns: YouBuyMilk, Friend1BuysMilk, and Friend2BuysMilk. Each routine must block if it is not their turn and must proceed to buy milk otherwise. Remember to write the initialization routines, and you may assume you always take the first turn. You can use pseudo-code for your solution.

(overflow sheet for your answers)