Department of Computer Science
University of Saskatchewan
CMPT 332
Midterm Exam

October 27, 2010	Name:_____
Number of Pages: 5	Signature:_____
Time: 50 minutes	Student Number: _____
Total: 50 marks.	NSID: _____

CAUTION - Candidates suspected of any of the following, or similar, dishonest practices shall be dismissed from the examination and shall be liable to disciplinary action.

1. Having at the place of writing any communication devices, any books, papers or memoranda, calculators, audio or visual cassette players, or other memory aid devices.
2. Speaking or communicating with other candidates.
3. Purposely exposing written papers to the view of other candidates. The plea of accident or forgetfulness shall not be received.

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	TOTAL
5	4	5	12	10	4	4	6	50

1. (5 marks) Assume you have 640 KB of main memory. Processes A, B, C, D, and E are presented to the medium-term scheduler for execution. How is memory allocated for these 5 processes in each of the following 2 memory management schemes? Show your answer by providing two diagrams of RAM after all the allocation and deallocation has been completed for worst fit algorithm.

Process	Size (in KB)
A	160
B	100
C	144
D	98
E	240

Algorithms for memory management:

Operations to perform:

- Allocate C,
- Allocate D,
- Allocate A,
- Free C,
- Allocate E,
- Allocate B,
- Free D,
- Allocate C.

IF the operations cannot be completed in the given order, indicate which operations would have to be aborted. Do not complete an aborted operation at a later time.

2. (4 marks). Semaphore operations are just another example of the critical section problem (i.e. they access their shared data in a mutually exclusive way). Why is it OK to use a software (or hardware) solution (such as the ones presented in class) to make the semaphore operations atomic while the use of software solutions for general critical section problems is frowned upon? (a few sentences max).

3. (5 marks) For the following Fork/Join code, draw the Process Flow Graph and give the Cobegin/Coend code (if possible) that corresponds to that particular code sequence.

```
     P1
     fork L2
     fork L3
     P5
     goto L8
L2:  P2
     fork L6
     P7
     goto L4
L3:  P3
L4:  join 2
     P4
     goto L8
L6:  P6
L8:  join 3
     P6
```

4. (12 marks) Process Scheduling. Consider the following job mix, including relative arrival times.

Job	Arrival Time	CPU Burst Time
A	0	10
B	3	6
C	5	8
D	4	4
E	6	5

Show the order of execution of these jobs using Round-Robin with a quantum of 3 and Shortest Job First. Use a time-line like in the interactive examples and/or class showing which jobs execute during which time slots. For RR, jobs are assumed to arrive just **before** the time slot indicated and are placed at the **back** of the ready queue.

```
RR

-------------------------------------------------------------------------
|        |
-------------------------------------------------------------------------
 0      3  etc...

TAT:                                          EFF:
```

```
SJF

-------------------------------------------------------------------------
|        |
-------------------------------------------------------------------------
 0

TAT:                                          EFF:
```

Indicate the avg. turnaround time (TAT) and the specific CPU efficiency for each algorithm on this set of tasks. For the efficiency, assume that each context switch takes .2 of a time unit. For the timeline, assume that each context switch takes 0 time. No calculator is necessary; leave the efficiency as a fraction.

5. (10 marks). Semaphores, Monitors, and Rendezvous IPC can all solve the same classes of problems, in that you can implement one facility with one of the others. Use S/R/R IPC to implement the Monitor primitives MonEnter, MonLeave, MonWait, and MonSignal as defined in class, and a MonitorServer thread. Provide detailed C-like pseudocode. Identify the List Operations that you need, but do not implement them. Assume that messages consist of a pointer to shared space and a buffer length. I've got you started.

```
int MonEnter(
{

}

int MonWait(
{

}

void MonServer (

}
```

```
int MonLeave(
{

}

int MonSignal(
{

}
```

6. (4 marks). What information (and in what order) must be saved at context switch time to adequately describe the state of a process? Be as complete and explicit as possible.

7. (4 marks) What performance issues are introduced when a multi-core processor schedules threads for execution? How do these issues interfere with each other.

8. (6 marks) Kernel-level threads and user-level threads provide the same interface/conceptual power to the application programmer. Internally, however, they are much different. Describe these differences in both the complexity of implementing kernel-level vs. user-level threads and the performance benefits/limitations that result from this choice.

———————- THE END———————