Department of Electrical Engineering, IIT Delhi

ELL405 Operating Systems: Minor II Examination

(Closed book/Closed Notes) Time: 1 hour Maximum Marks: 25

"Thou shalt not covet thy neighbour's answers"

1. **Optimality of SJF: Attempt this question First? ;-)** Given n jobs at time instant zero. SJF schedules them shortest-job-first. Assume that SJF schedules a certain job i before job j, the time for the jobs being T_i and T_j, respectively, $i + 1 \leq j \leq n$. Prove that SJF is optimal by the Method of Contradiction (*reductio ad absurdum*) as follows: assume that there is an optimal scheduling policy which puts job j before job i, and has a smaller average waiting time. (The position of all other jobs in the scheduling strategies remains the same.) Show that this leads to a contradiction. Give a very general proof, by considering the situation for a job k, which lies between the positions of i and j. *(5 marks)*

2. **Set for the Test? Read carefully, then Write!** Consider the readers and writers problem as done in class, with a bias towards readers. When a writer is writing, no other writer, or reader can access the database. When readers are reading, no writer can write to the datbase. More than one reader can access the database concurrently. You have to implement this with the `TestAndSet()` instruction, not semaphores/mutexes.

```
Boolean TestAndSet(Boolean *target)
{ Boolean rv = *target;
  *target = TRUE;
  return rv; }
```

You are given `Boolean` variables `wrt` and `mutex`, and a shared variable `readcount` (the number of readers). Assume `FALSE` and `TRUE` to be 0 and 1, respectively. Take care to make suitable initialisations. *(11 marks)*

3. **Brevity is the Sole Soul of Wit** *(1+2+2+2+2 marks)*

   (a) Real-time tasks are considered periodic. Why?

   (b) How does an Intel dual-core processor schedule two threads?

   (c) What is the minimum granularity of Linux's CFS?

   (d) Why does CFS end up making an I/O-bound task run more often?

   (e) If a semaphore on a uniprocessor (one single core processor CPU) were not implemented in conjunction with the OS's CPU scheduler (the block and wake-up operation associated with a `wait()` and `signal()` operation), what would be the main problem associated with the following plain-vanilla implementation of semaphores?

`wait(S) {while(S<=0); S--;}`	`signal(S) {S++;}`