ASSIGNMENT 3

1. If a multithreaded process forks, a problem occurs if the child gets copies of all the parent's threads. Suppose that one of the original threads was waiting for a keyboard input. Now two threads are waiting for the keyboard input, one in each process. Does this problem ever occur in single-threaded processes? [1]
2. Is the register set a part of the per-thread or per-process state? Why? [1]
3. Let us consider the following system: An average process runs for a time T before it blocks on I/O. A context switch takes time S, which is essentially wasted. For round robin scheduling with quantum Q, give a formula for CPU efficiency, which is (time when CPU is doing something useful/total time) for each of the following cases: [2.5]
   a. $Q = \infty$
   b. $Q > T$
   c. $S < Q < T$
   d. $Q = S$
   e. $Q \to 0$
4. If P() and V() are not executed atomically, then do we still get a correct solution to the critical section problem? Are any conditions, i.e. mutual exclusion, progress and bounded waiting, violated? If yes, how? If not, then show that it holds. [2]
5. Given a set of processes and their job completion times, prove that the Shortest Job First algorithm gives the shortest waiting time. [1.5]
6. In the following program:

```
shared int N = 50, tally; /* shared across multiple processes */
void total() {
   int count;
   for(count=1; count < N; count++) {
      tally++;
   }
}
void main() {
   pid_t pid = fork();
   tally = 0;
   if (pid == 0) { /* shares the variable tally and N */
      total();
   } else if (pid > 0) {
      total();  /* shares tally and N with child process */
   }
}
```

Determine the proper upper and lower bound on the final value of the shared variable *tally*. Assume that processes can run at any speed, and a value can only be incremented after it has been loaded into a register by a separate machine instruction. (Hint: lower bound is not 50) [2]