Concurrency: Mutual Exclusion and Synchronization

5.1 MUTUAL EXCLUSION: SOFTWARE APPROACHES

1. Define the followings:
    a. Mutual exclusion
    b. Critical section
2. True or false:
    a. Peterson's algorithm can not be generalized to the case of processes

5.2 PRINCIPLES OF CONCURRENCY

3. Define the followings:
    1. Race condition

4. What are the requirements for mutual exclusion

5.3 MUTUAL EXCLUSION: HARDWARE SUPPORT

5. Machine-instruction approach that enforces mutual exclusion has a number of disadvantages, mention them.

5.4 SEMAPHORES

6. Consider the following definition of semaphores:

```
void semWait(s)
{
    if (s.count > 0) {
      s.count--;
    }
    else {
      place this process in s.queue;
      block;
    }
}
void semSignal (s)
{
    if (there is at least one process blocked on
        semaphore s) {
        remove a process P from s.queue;
        place process P on ready list;
    }
    else
        s.count++;
}
```

Figure 1: Definition 1 of semaphores

```
struct semaphore {
       int count;
       queueType queue;
};
void semWait(semaphore s)
{
       s.count--;
       if (s.count < 0) {
         /* place this process in s.queue */;
         /* block this process */;
       }
}
void semSignal(semaphore s)
{
       s.count++;
       if (s.count<= 0) {
         /* remove a process P from s.queue */;
         /* place process P on ready list */;
       }
}
```

Figure 1: Definition 2 of semaphores

Compare the two definitions of semaphores. Note one difference: With the preceding definition, a semaphore can never take on a negative value. Is there any difference in the effect of the two sets of definitions when used in programs? That is, could you substitute one set for the other without altering the meaning of the program?

General questions

7. Consider the following program:

```
const int n = 50;
int tally;
void total() {
   int count;
   for (count = 1; count <= n; count++) {
         tally++;
   }
}
void main() {
   tally = 0;
   parbegin (total (), total ());
   write (tally);
}
```

   a. Determine the proper lower bound and upper bound on the final value of the shared variable tally output by this concurrent program. Assume processes can execute at any relative speed and that a value can only be incremented after it has been loaded into a register by a separate machine instruction.
   b. Suppose that an arbitrary number of these processes are permitted to execute in parallel under the assumptions of part (a). What effect will this modification have on the range of final values of tally?

8. Consider the following program:

```
boolean blocked [2];
int turn;
void P (int id) {
   while (true) {
         blocked[id] = true;
         while (turn != id) {
               while (blocked[1-id])
               /* do nothing */;
               turn = id;
         }
         /* critical section */
         blocked[id] = false;
         /* remainder */
   }
}
void main() {
   blocked[0] = false;
   blocked[1] = false;
   turn = 0;
   parbegin (P(0), P(1));
}
```

This software solution to the mutual exclusion problem for two processes is proposed. Find a counterexample that demonstrates that this solution is incorrect.