CSE 450 Operating System

Homework on synchronization

Name: ____partial solution______

Question 1. [10 pts] Answering the following briefly regarding counting semaphore. Give a justification for each answer, [which means a simple yes or no answer would not receive full marks.]

- Can there be a thread blocked on a semaphore with non-negative value?
- Can a semaphore have a negative value without having any threads blocked on it?

According to the implementation of semaphore,

```
wait (S){
        value--;
        if (value < 0) {
                add this process to waiting queue
                block();
        }
}
Signal (S){
        value++;
        if (value <= 0) {
                remove a process P from the waiting queue
                wakeup(P);
        }
}
```

if the value of the semaphore is 0 (non-negative), then the thread calls wait(S) will be blocked; if the value of the semaphore is positive, then any thread call wait(S) will result the value of semaphore >= 0, which means the calling thread would enter the critical section.

Recall the pattern of using semaphore of any thread as wait (S) -> C.S. -> Signal (S). We assume the initial value for a counting semaphore is positive, then if the semaphore has a negative value, then there must be threads blocked on it after calling Wait (S); otherwise, there would be no Wait (S) calls preceding this, and the value of semaphore won't go to negative.

Question 2. [20 pts] In the following code, four processes produce output using the routine "printf" and synchronize using three semaphores "R", "S" and "T." We assume function 'printf' wont cause context switch.

Semaphore R=1, S = 3, T = 0; /* initialization */

```
/* process 1 */        /* process 2 *         /* process 3 */       /*process 4 */
while(true) {          while(true) {          while(true) {         while(true) {
   P(S);                  P(T);                  P(T);                 P(R);
   printf('A');           printf ('B');          printf ('D');         printf ('E');
                          printf ('C');          V(R);                 V(T);
                          V(T);               }                     }
}                      }
```

a) How many A and B's are printed when this set of processes runs?

3 A, any number of B

b) What is the smallest number of D's that might be printed when this set of processes runs?

0 (1): there is a chance the process 3 never gets a chance to run, so the min number of D is 0; if you argue that every process will eventually get chance to run, then the minimum number of "D" printed is 1.

c) Is AEBCBCDAA a possible output sequence when this set of processes runs? Clarify your answer.

Yes, the running order of processes could be p1->p4->p2->p2->p3->p1->p1 to produce the above output sequence.

Question 3. [**Critical Section: 20 pts**] Consider the following two processes P[i] and P[j]. Initially, flag[i] = flag [j] = false.

```
Do{                                  Do{
   flag[i]=true;                        flag[j]=true;
   While(flag[j]);                      While(flag[i]);

   critical section                     critical section

   flag[i] = false;                     flag[j] = false;

   remainder section                    remainder section
} while(1);                          } while(1);
```

a) Does the above program satisfy the “progress” requirement? Justify your answer with an informal proof or counterexample. [Simple “Yes” or “No” without explanation]

No. When both processes have their flags set to be true, then they will do busy waiting at the same time.

b) Is mutual exclusion assured? Justify your answer with an informal proof or counterexample.

Yes. When one process in the critical section, its flag will be true until leaving, which makes the other process do busy waiting.