CSI 4500 Operating System

Homework on synchronization

Name: ____partial solution_____

Question 1. [25 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 */
while(true) {
   P(S);
   printf('A');
}
```

```
/* process 2 *
while(true) {
   P(T);
   printf ('B');
   printf ('C');
   V(T);
}
```

```
/* process 3 */
while(true) {
   P(T);
   printf ('D');
   V(R);
}
```

```
/*process 4 */
while(true) {
   P(R);
   printf ('E');
   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 2. [**Critical Section: 25 pts**] Consider the following two processes P[i] and P[j]. Initially, flag[i] = flag [j] = false.

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

   critical section

   flag[i] = false;

   remainder section
} while(1);
```

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

   critical section

   flag[j] = false;

   remainder section
} 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.

Question 4. [**Process Synchronization: Bonus 10 pts**] The Sleeping Barber Problem: A barbershop consists of a waiting room with **n** chairs and the barber room containing the barber chair. If there are no customers to be served, the barber goes to sleep. If a customer enters the barbershop, and all chairs are occupied, then the customer leaves the shop. If the barber is busy but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber. Write a program with **semaphores** to coordinate the barber and the customers.

Semaphore Declaration:

```
customers_available = 0;
barber_available = 0;
mutex = 1;
waiting = 0;
```

Barber Process

```
while(True) {

   p(customers_available);
   p(mutex);
      waiting--;
   v(mutex);
   v(barber_available);
   //cut hair();




}
```

Semaphore Declaration:

Customer Process

```
while(True) {


   p(mutex)
   if(waiting<n) {
      waiting++;
       v(mutex);
       v(customers_available);
       p(barber_available);
       //get_haircut();
   else
       v(mutex);


}
```