CS3600 – Worksheet02 Synchronization

Problem 1 & 2 is to understand monitor better, Problem 3 to understand the classic dining philosopher problem synchronized using semaphore.

1. While a process is executing inside the function A of the monitor m, the following calls are issued by different processes:
   m.A(); m.B(); m.B(); m.B(); m.B(); m.A(); m.A()

```
monitor m {
            int x = 10, y = 2
            condition c
            A(){
(1)           x++
(2)           c.signal
(3)           y = x - 2 }
            B(){
(4)           if (x > 10)
(5)                x--
(6)           else{c.wait
(7)                x--} }
```

   Using the line numbers in the code, trace the sequence of instruction execution. Show any changes of x and y at the end of each instruction.

   initially x = 10 y = 2

   1. x = 11 y = 9 from line 1 & 2
   2. x = 10 y = 9 from line 4
   3. x = 10 y = 9 from line 4
   4. x = 10 y = 9 from line 4
   5. x = 10 y = 9 from line 4
   6. x = 11 y = 9 from line 1
   7. x = 10 y = 9 from line 6 & 7
   8. x = 11 y = 8 from line 1 & 2 & 3

2. Complete the code below for implementing producer- consumer problem using a monitor.

```
monitor ProducerConsumer{
     condition full, empty;
     int count;

      enter();
      {
        if (count == N) wait(full);
        put_item(item);
        count = count + 1;
        if (count == 1) signal(empty);
      }

      remove();
      {
        if (count == 0)  wait(full);
        remove_item(item);
        count = count - 1;
        if (count == N-1) signal(empty);
      }

      count = 0;
   }
   Producer();
   {
     while (TRUE)
```

```
 {
    make_item(item);
    ProducerConsumer.enter();
 }
}
Consumer();
{
  while (TRUE)
  {
    ProducerConsumer.remove();
    consume_item;
 }
}
```

3. Trace the Dining Philosopher problem and write the flow of states using table below. If philosophers 2 , 3 and 5 try to eat.

```
  me:    semaphore, initially 1;                    /* for mutual exclusion */
  s[5]:  semaphore s[5], initially 0;               /* for synchronization */
  pflag[5]: {THINK, HUNGRY, EAT}, initially THINK;  /* philosopher flag */
procedure philosopher(i)
  {
while TRUE do
 {
 THINKING;
 take_chopsticks(i);
 EATING;
 drop_chopsticks(i);
 }
}
procedure take_chopsticks(i)
{
  P(me);               /* critical section */
  pflag[i] = HUNGRY;
  test[i];
  V(me);                 /* end critical section */
  P(s[i])              /* Eat if enabled */
}

void test(i)            /* Let phil[i] eat, if waiting */
{
if ( pflag[i] == HUNGRY && pflag[i-1] != EAT && pflag[i+1] != EAT)then
  {
    pflag[i] = EAT;
    V(s[i])
   }
}
void drop_chopsticks(int i)
{
  P(me);                /* critical section */
  pflag[i]=THINKING;
  test(i-1);               /* Let phil. on left eat if possible */
  test(i+1);               /* Let phil. on right eat if possible */
  V(me);                  /* up critical section */
}
```

Write the change in state of philosophers [T,E,H] . Initially all are in the thinking state[T].

P[1]	P[2]	P[3]	P[4]	P[5]
T	T	T	T	T
H	T	T	H	T

CS3600 – Worksheet02 Synchronization

E	H	T	H	E
T	E	T	E	T
T	T	H	T	T
H	T	E	T	H
E	T	T	T	E
T	H	T	H	T
T	E	T	E	T
H	T	H	T	H
E	T	H	T	E