# Concurrency: Mutual Exclusion and Synchronization

## 5.4 cont + 5.5 MONITORS

**1.** Consider the following program:

```
P1: {
    shared int x;
    x = 10;
    while (1) {
          x = x - 1;
          x = x + 1;
          if (x != 10){
             printf("x is %d",x)
          }
    }
}
```

```
P2:{
      shared int x;
      x = 10;
      while ( 1 ) {
            x = x - 1;
            x = x + 1;
            if (x!=10){
            printf("x is %d",x)
            }
      }
}
```

Note that the scheduler in a uniprocessor system would implement pseudo parallel execution of these two concurrent processes by interleaving their instructions, without restriction on the interleaving order.

a. Show a sequence (i.e., trace the sequence of interleaving of statements) such that the statement "x is 10" is printed.
b. Show a sequence such that the statement "x is 8" is printed. You should remember that the increment/decrements at the source language level are not done atomically, i.e., the assembly language code:

```
LD R0,X /* load R0 from memory location x */
INCR R0 /* increment R0 */
STO R0,X /* store the incremented value back in X */
```

implements the single C increment instruction (x = x + 1).

**2.** Show where/how to add semaphores to the program in the preceding problem to insure that the printf() is never executed. Your solution should allow as much concurrency as possible.

**3.** Considering wait and signal operations, what is the difference between semaphores and monitors?

## 5.6 MESSAGE PASSING

**4.** What is massage passing? How can it be used to enforce mutual exclusion?

## 5.7 READERS/WRITERS PROBLEM

**5.** Explain what is the problem with this implementation of the one-writer many readers problem?

```
int readcount; // shared and initialized to 0
Semaphore mutex, wrt; // shared and initialized to 1;
// Writer:                                  // Readers:
semWait(wrt);                               semWait(mutex);
/* Writing performed*/                            readcount := readcount + 1;
semSignal(wrt);                             if readcount == 1 then semWait(wrt);
                                            semSignal(mutex);
                                            /*reading performed*/
                                            semWait(mutex);
                                            readcount := readcount - 1;
                                            if readcount == 0 then semSign
                                          (wrt);
                                            semSignal(mutex);
```

**6.** Write the pseudo code for solving one writer and many readers problem using monitor.

## General questions

**7.** Consider Lamport's Bakery Algorithm

```
boolean choosing[n];
int number[n];
while (true) {
    choosing[i] = true;
    number[i] = 1 + getmax(number[], n);
    choosing[i] = false;
    for (int j = 0; j < n; j++){
      while (choosing[j]) { };
      while ((number[j] != 0) && (number[j],j) < (number[i],i))
             { } ;
      }
    /* critical section */;
    number [i] = 0;
    /* remainder */;
}
```

The arrays choosing and number are initialized to false and 0, respectively. The $i^{th}$ element of each array may be read and written by process i but only read by other processes.

The notation (a, b) < (c, d) is defined as:

$$(a < c) \text{ or } (a = c \text{ and } b < d)$$

a. Describe the algorithm in words.

b. Show that this algorithm avoids deadlock.

**Extra credit:** c. Prove that the algorithm enforces mutual exclusion.