Sheet4

CONCURRENCY: MUTUAL EXCLUSION

1) List four design issues for which the concept of concurrency is relevant.
2) What are three contexts in which concurrency arise?
3) What is the basic requirement for the execution of concurrent processes?
4) List three degrees of awareness between processes and briefly define each.
5) What is the distinction between competing processes and cooperating processes?
6) List the three control problems associated with competing processes and briefly define each.
7) List the requirements for mutual exclusion.
8) What operations can be performed on a semaphore?
9) What is the difference between binary and general semaphores?
10) What is the difference between strong and weak semaphores?
11) At the beginning of Section 5.1, it is stated that multiprogramming and multiprocessing present the same problems, with respect to concurrency. This is true as far as it goes. However, cite two differences in terms of concurrency between multiprogramming and multiprocessing.
12) 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 order of the interleaving.

a) Show a sequence (i.e., trace the sequence of interleavings 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, that is, 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).

13) 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?

14) Is busy waiting always less efficient (in terms of using processor time) than a blocking wait? Explain.

15) 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.

16) A software approach to mutual exclusion is Lamport's bakery algorithm [LAMP74], so called because it is based on the practice in bakeries and other shops in which every customer receives a numbered ticket on arrival, allowing each to be served in turn. The algorithm is as follows:

```
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) or (a = c and b < d)

a) Describe the algorithm in words.
b) Show that this algorithm avoids deadlock.
c) Show that it enforces mutual exclusion.

17) Now consider a version of the bakery algorithm without the variable choosing. Then we have

```
1 int number[n];
2 while (true) {
3    number[i] = 1 + getmax(number[], n);
4    for (int j = 0; j < n; j++){
5      while ((number[j] != 0) && (number[j],j) < (number[i],i)) { };
6    }
7    /* critical section */;
8    number [i] = 0;
9    /* remainder */;
10 }
```

Does this version violate mutual exclusion? Explain why or why not.

How to submit the homework assignments?

- Solve the sheet individually without looking up the solution on the Internet. The sheet is to practice; it is a learning tool not an exam.
- Assignments are to be **handwritten**.
- Papers are to be scanned (I like camscanner app). Put all images in a pdf file (camscanner does that for you)
- Use MS Teams to submit
    - Your filename should be your user id