**Comp 4735 Winter 2015**

## Lab Instructor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ SET : 4D

**Name: Jens Christiansen Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# Lab 4

Solve the following exercises. Work in pairs. Discuss each exercise with your lab instructor.

1. Describe, compare and contrast the concepts presented in Table 5.1.
2. Discuss the concept of race condition (5.1) and give a code example.

A race condition is a situation where the outcome is determined by the speed in which various threads or processes manipulate shared data. A race condition is a stochastic system in which the designer must implement proper concurrency mechanisms through techniques such as the use of semaphores, mutexes, and monitors. An example of a race condition is where two processes are incrementing a shared variable x = 0. If Process 1 (P1) loads the variable at the same time as Process 2 (P2) loads it, then both processes will have x = 0. When they each increment and then store x back into memory, x will hold the value of 1, as opposed to 2, as we might expect when the two Processes increment x.

|  |  |
| --- | --- |
| Process 1 {  Shared int x;  x++;  printf(“x is %d”, x);  } | Process 2 {  Shared int x;  x++;  printf(“x is %d”, x);  } |

The following time slices may occur, which would result in an unexpected output:

P1: Load x into register; // x = 0

P2: Load x into register; // x = 0

P1: Increment x; // x = 1

P2: Increment x; // x = 1

P1: Store x back into memory; // x = 1

P2: Store x back into memory; // x = 1

P1: printf(“x is %d”, x); // “x is 1”

1. Discuss the concept of mutual exclusion.

Mutual exclusion is a concept where a process or thread has exclusive access to a shared resource. When the process or thread accesses this resource, it “locks” its availability, and no other process or thread can manipulate that resource. Once the process has finished using the resource, it will “unlock” it for other processes or threads to access.

Requirements for Mutual Exclusion

* Must be enforced
* A process that halts must do so without interfering with other processes
* No deadlocks or starvation
* A process must not be denied access to a critical section when there is no other process using it
* No assumptions are made about relative process speeds or number of processes
* A process remains inside its critical section for a finite time only
* There are ways in which the requirements for mutual exclusion can be satisfied:
  + Hardware approach using special-purpose machine instructions
  + Software approach

1. Solve problem 5.3 in textbook.
   1. Sequence:
      1. P1: x = 10; // 10
      2. P2: x = 10; // 10
      3. P1: while (1)
      4. P2: while (1)
      5. P1: x = x – 1; // 9
      6. P1: x = x + 1; // 10
      7. P2: x = x – 1; // 9
      8. P1: if (x != 10) // true
      9. P2: x = x + 1; // 10
      10. P1: printf(“x is %d”, x); // “x is 10”
   2. Sequence:
      1. P1: x = 10; // 10
      2. P2: x = 10; // 10
      3. P1: while (1)
      4. P2: while (1)
      5. P1: x = x – 1; // 9
      6. P2: LD R0,X // 9
      7. P2: DEC R0 // 8
      8. P1: x = x + 1; // 10
      9. P2: STO R0; // 8
      10. P1: if (x != 10) // true
      11. P1: printf(“x is %d”, x) // “x is 8”
2. What is mutual exclusion hardware support? What are the advantages and disadvantages of this method?

There are several approaches to mutual exclusion:

**Interrupt Disabling**

If a resources needs to be locked, the system will create a critical section by disabling all interrupts, effectively preventing other processes from interrupting the current processes. Although this technique guarantees mutual exclusion, the process will not be able to interleave processes. Furthermore, it will only work with a uniprocessor system, and not work in a multiprocessor architecture: More than one process can be running on each processor at the same time that can interrupt each other, breaking the mutual exclusion guarantee.

**Special Machine Instructions**

1. What is a semaphore?
   1. Is the incrementation or decrementation of the counter variable safe?
   2. Is the OS aware of the semaphore?
   3. Write the pseudo-code of a semaphore’s methods. Any race condition?
2. Consider a doctor’s office with two doctors each in his room. Consider a waiting room with four chairs. If no chair is available, patients should wait outside. Solve the problem in pseudo-code with semaphores.