# Homework assignment 2

T-106.420 Concurrent Programming

Family name: **Chaparro González**
First name: **Diego**
Student number: **59881P**
dchaparro@acm.org

1st November 2002

# Contents

# 1 Exercise

**Consider the following fine-grained synchronization program of two processes**

```
int turn1 = 0, turn2 = 0;
P1::
while (true) {
        turn1 = turn2 + 1;
        while (turn2 != 0 && turn1 > turn2)
                skip;
        critSection(1); turn1 = 0; nonCritSection(1);
}
P2::
while(true) {
        turn2 = turn1 + 1;
        while (turn1 != 0 && turn2 > turn1)
                skip;
        critSection(2); turn2 = 0; nonCritSection(2);
}
```

**The processes can reach their critical sections at the same time.**

## 1.1 a) Explain how.

It can happen because the instructions:

```
turn1 = turn2 + 1;
```

and

```
turn2 = turn1 + 1;
```

aren't atomic, and these are the source of the problem.

For example, the first process starts to run and it's executing the instruction *turn1 = turn2 + 1*. First load a register with turn2 (which value is 0), but before it adds 1 to the register, the second process starts to run. This second process starts to execute the instruction *turn2 = turn1 + 1*, read the value of x and it's 0 yet, because the process 1 hasn't written it yet.

Then the two process follow their execution and finish the execution of these two instructions (*turn1=turn2+1* and *turn2=turn1+1*) with the values

- turn1 == 1
- turn2 == 1

With these values both processes can reach their critical sections because the expresion *(turn*1 > *turn*2*)* and *(turn*2 > *turn*1*)* are false respectively.

## 1.2 b) Develop a working busy-waiting solution based on this program

This can be one working solution:

```
int turn1 = 0, turn2 = 0;
P1::
while (true) {
        <turn1 = turn2 + 1;>
        <await (turn2 == 0 or turn1 <= turn2)>
        critSection(1); turn1 = 0; nonCritSection(1);
 }
 P2::
 while (true) {
        <turn2 = turn1 + 1;>
        <await (turn1 == 0 or turn2 <= turn1)>
        critSection(2); turn2 = 0; nonCritSection(2);
}
```

# 2 Exercise

**Suppose your machine has to following atomic instruction:**

```
flip(lock): <lock = (lock + 1) % 2; # flip the lock
             return (lock);>
```

**Someone suggests the following solution to the critical section problem for two processes:**

```
int lock = 0;
process CS[i = 1 to 2]{
        while (true) {
                while (flip(lock) != 1)
                        while (lock != 0) skip;
                critSection();
                lock = 0;
                nonCritSection();
        }
}
```

## 2.1 a) This solution does not work. Give an execution order that results in both processes being in their critical sections at the same time

Lock is 0.

Process 1 starts to execute, and enter in the critical section. The lock is 1.

Then the process 2 start to execute, the *flip(lock)* returns 0, which is different from 1 and enters in the inner while loop. Now lock is 0. The condition of the inner loop *(while (lock != 0)* is false, so it returns to the outer loop. Now the flip(lock) returns 1, so the while condition (flip(lock) != 1) is false, and the second process enters into the critical section while process 1 is still there.

## 2.2 b) Suppose that the first line in the flip instruction is changed to do addition modulo 3 rather than modulo 2. Will the solution now work for two processes, and if so, is the solution fair? Explain.

Yes, with this change it works because in the above case, when the second process enters in the inner loop (*while (lock!=0)*)the value of lock is 2, and this process enters in this loop until the process 1 change the value of lock to 0 when it exits from the critical section.

In this solution we have an example of Test-and-Set instructions, so the property of "eventual entry" is not guaranteed, because the lock will become false infinitely often for process 2, and for this reason this maybe it's not a fair solution, but it works.