Universitetet i Oslo
Institutt for Informatikk
PMA
Olaf Owe, Martin Steffen

INF 4140: Models of Concurrency

Høst 2015 **Series 2** 7. 9. 2015

Topic: About Chap. 2& 3: synchronization, critical sections

Issued: 7. 9. 2015

`Exercises: 2.17, 2.18, 2.33, 3.1, 3.7, 3.8 from the textbook`

Exercise 1 (2.17)

```
co
    <await (x ≥ 3) x := x−3>
  ||
    <await (x ≥ 2) x := x−2>
  ||
    <await (x = 1) x := x+5>
oc
```

For which initial values does the program terminate (under weak scheduling). What are the corresponding final values. Explain the answer.

Exercise 2 (2.18)

```
co
   <await (x > 0) x := x−1;>
||
   <await (x < 0) x := x+2;>
||
   <await (x = 0) x := x−1;>
oc
```

For which initial values does the program terminate (under weak scheduling). What are the corresponding final values. Explain the answer.

Exercise 3 (2.33)

```
int x:=10; c:= true;


co
   <await x = 0>;  c := false

||
     while (c) < x := x-1>

oc
```

1. Termination under weak fairness?
2. Termination under strong fairness?
3. Add the following statement as 3rd arm of the `co`-statement:

```
      while (c) {if (x < 0)    <x := 10> ;}
```

Exercise 4 (Dekker's algo (3.1)) The code shows the initialization and process P_1, a second P_2 is symmetric.

```
bool enter1 = false, enter2 = false;
int turn = 1;

process P1{
  while (true){

    enter1 := true                  ## entry protocol
    while(enter2){
      if(turn = 2){
        enter1 := false;
        while(turn = 2) skip;
        enter1 := true;
      }
    }

    CS;

    enter1 := false;                ## exit protocol
    turn := 2;
    non-CS;
  }
}
```

1. mutex?
2. deadlock
3. unnecessary delay
4. eventual entry

Also: how many times can one process that wants to enter its critical section be bypassed by the other before the first gets in?

Exercise 5 (3.7) Consider the following code snippet (due to Lamport [?])

```
int lock = 0;
process CS[i = 1 to n]{
  while(true){
    <await (lock = 0)>;
    lock := i;
    Delay
        while(lock != i){
          <await (lock = 0)>; lock := i; Delay;
    p     }
    }
    CS;
    lock := 0;
    non-CS;
}
```

1. Suppose the delay code is deleted.
   (a) mutex?
   (b) deadlock?
   (c) unnecessary delay?
   (d) eventual entry
2. Suppose the Delay code is added and long enough. Reconsider your answers under that circumstances.

Exercise 6 (3.8) Consider the following code. Not that the flip-operation is assumed to be atomic (for instance, representing a HW operation). Then consider the sketched code intended to solve the CS problem.

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

int lock = 0;                      # shared variable

process CS[i = 1 to 2]{
  while(true){
   while(flip(lock) != 1)
     {while(lock != 0) skip;}
   CS;
   lock := 0;
   non-CS;
  }
}
```

1. Spot the defect in the code, violating the basic safety assumption, i.e., “mutual exclusion”.
2. What happens if the calculation is done modulo 3, instead of modulo 2 as now?