T–79.4301 **Spring 2006**

Parallel and Distributed Systems

Tutorial 1 - Wed Feb 1, 2006 11:15 and Fri Feb 3, 2006 14:15

Note: To get started, this tutorial is a demonstration round. The next exercises can also contain a mixture of demos and regular tutorial exercises.

1. Dekker's mutual exclusion algorithm (1962) tries to ensure that at most one of two processes is in a critical section at any time. The algorithm is described by the following pseudo-code:

```
// Dekker's mutex algorithm, two parallel processes 0 and 1

var flag: array[0..1] of boolean;
var turn: 0;

// flag is initialized to all false,
// and turn has the initial value 0
// The algorithm for process i then becomes:

// Infinite loop
while true do
     // [noncritical section]

     // i is my index, j is the other process
     i := mypid(); j=1-mypid();
     flag[i] := true;
     // [trying section]
     while flag[j] do
          if turn = j then
                    begin
                         flag[i] := false;
                         while turn = j do idle enddo;
                         flag[i] := true;
                    end;
          endif;
     enddo;

     // [critical section]
     turn := j;
     flag[i] := false;
enddo;
```

   a) Model Dekker's algorithm in Promela, the input language of the model checker Spin (`http://www.spinroot.com/`).

b) Add an assertion mechanism into the code which triggers when both processes are in the critical section at the same time.

c) Check with Spin whether Dekker's algorithm guarantees mutual exclusion for two processes.

d) In 1980 Doran and Thomas suggested the following simpler variant of the inner loop:

```
if flag[j] then begin
     if turn = j then
          begin
               flag[i] := false;
               while turn = j do idle enddo;
               flag[i] := true;
          end;
     endif;
     end;
endif;
```

Does the algorithm work after this modification?
(Hint: Use Spin.)

e) Does Peterson's algorithm (below) guarantee mutual exclusion?

```
// Peterson's mutex algorithm, two parallel processes 0 and 1
var flag: array[0..1] of boolean;
var turn: 0;
// flag is initialized to all false,
// and turn has the initial value 0
// The algorithm for process i then becomes:

// Infinite loop
while true do
     // [noncritical section]

     // i is my index, j is the other process
     i := mypid(); j=1-mypid();
     flag[i] := true;
     // [trying section]
     turn := i;
     while (flag[j] = true and turn = i) do idle enddo;

     // [critical section]

     flag[i] := false;
enddo;
```