Concurrent and Distributed Systems

Mitre Flavia Antonia

November 17, 2019

Third Year
Group: C.E.N. 3.1
Lecturer: Costin Bădică
Teaching assistant: Cristinel Ungureanu

1 Problem statement

1.1 First problem

Implement Dekker algorithm using 2 processes (as available in Critical Sections Laboratories).

Assignment: Implement the Dekker algorithm in Promela, describe the output and different scenarios (in multiple executions)

Dekker's algorithm	
boolean wantp ← false, boolean wantq ← false, integer turn ← 1	
p	q
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: wantp ← true	q2: wantq ← true
p3: while wantq = true	q3: while wantp = true
p4: if turn = 2	q4: if turn = 1
p5: wantp ← false	q5: wantq ← false
p6: await turn = 1	p6: await turn = 2
p7: wantp ← false	p7: wantq ← false
p8: critical section	q8: critical section
p9: turn ← 2	q9: turn ← 1
p10: wantp ← false	q10: wantq ← false

2 Implementation

2.1 First problem

For solving the first problem I've created:

- the global variables wantp, wantq, turn
- two active proctypes: P and X.
  The variables wantp(first proctype) and wantq(second proctype) name what thread is already in execution.

```
1. wantp ← false
2. wantq ← false
3. turn ← 1
2. active proctype P()
3.     do
4.     wantp ← true
5.         do
6.         wantq →
7.             if turn = 2 do
8.                 wantp ← false
9.                 turn ← 1
10.                wantp ← true
11.            end if
12.        else → break
13.        end do
14.        turn ← 2
15.        wantp ← false
16.    end do
17. end proctype
18. active proctype X()
19.     do
20.     wantq ← true
21.         do
22.         wantp →
23.             if turn = 1 do
24.                 wantq ← false
25.                 turn ← 2
26.                 wantp ← true
27.             end if
28.         else → break
29.         end do
30.         turn ← 1
31.         wantp ← false
32.    end do
33. end proctype
```

3 Experimental Data

3.1 First problem

```
  0:    proc  - (:root:) creates proc  0 (P)
  0:    proc  - (:root:) creates proc  1 (X)
1 X     27  wantq = 1
Process Statement         wantq
1 X     29  else          1
0 P     7   wantp = 1     1
Process Statement         wantp     wantq
1 X     40  turn = 1      1         1
1 X     41  wantq = 0     1         1
0 P     9   else          1         0
0 P     20  turn = 2      1         0
Process Statement         turn      wantp     wantq
0 P     21  wantp = 0     2         1         0
1 X     27  wantq = 1     2         0         0
0 P     7   wantp = 1     2         0         1
1 X     29  wantp         2         1         1
0 P     9   wantq         2         1         1
1 X     31  else          2         1         1
0 P     11  turn==2       2         1         1
1 X     29  wantp         2         1         1
0 P     13  wantp = 0     2         1         1
1 X     31  else          2         0         1
1 X     29  else          2         0         1
1 X     40  turn = 1      2         0         1
0 P     14  turn==1       1         0         1
0 P     15  wantp = 1     1         0         1
1 X     41  wantq = 0     1         1         1
1 X     27  wantq = 1     1         1         0
-------------
depth-limit (-u40 steps) reached
#processes: 2
 40:    proc  1 (X) dekker.pml:29 (state 12)
 40:    proc  0 (P) dekker.pml:9 (state 12)
2 processes created
```

Figure 1: First output

The first proctype executed is X. The value of wantq changes from false to true, the same for wantp. Then turn becomes 1 (it's value does not actually change) and wantq becomes false. Turn becomes 2 (changes value) and wantp becomes false. Because turn equals 2, wantp (from proctype P) becomes false. The two else conditions (proctype X line 29 and proctype X line 31 do nothing that affects the execution because turn equals 2, not 1). Turn equals 2, so, in proctype P (line 11) wantp becomes false (line 13). Wait until turn becomes 1 again. The same for the two elses as above. Turn becomes 1 and wantp becomes true. Wantq becomes 0 and then 1, affecting nothing. This were the 40 steps.

```
  0:    proc  - (:root:) creates proc  0 (P)
  0:    proc  - (:root:) creates proc  1 (X)
1 X     27  wantq = 1
Process Statement          wantq
1 X     29  else           1
0 P     7   wantp = 1      1
Process Statement          wantp      wantq
0 P     9   wantq          1          1
1 X     40  turn = 1       1          1
1 X     41  wantq = 0      1          1
1 X     27  wantq = 1      1          0
0 P     11  else           1          1
1 X     29  wantp          1          1
1 X     31  turn==1        1          1
1 X     33  wantq = 0      1          1
0 P     9   else           1          0
-------------
depth-limit (-u20 steps) reached
#processes: 2
 20:    proc  1 (X) dekker.pml:34 (state 5)
 20:    proc  0 (P) dekker.pml:20 (state 15)
2 processes created
```

Figure 2: Second output

The first proctype executed is X. Wantq becomes true, the else on line 29 has no effect (because wantp is false). Wantp becomes true. Turn becomes 1 and wantq becomes 0. Wantq becomes 1. The else on line 11 has no effect, because turn equals 1. Wantp is true(line 29) and turn is 1 (line 31) so wantq becomes false (the critical section of proctype X). The last else has no effect.

```
  0:    proc  - (:root:) creates proc  0 (P)
  0:    proc  - (:root:) creates proc  1 (X)
0 P     7   wantp = 1
Process Statement          wantp
1 X     27  wantq = 1      1
Process Statement          wantp      wantq
1 X     29  wantp          1          1
1 X     31  turn==1        1          1
1 X     33  wantq = 0      1          1
0 P     9   else           1          0
0 P     20  turn = 2       1          0
Process Statement          turn       wantp      wantq
0 P     21  wantp = 0      2          1          0
1 X     34  turn==2        2          0          0
0 P     7   wantp = 1      2          0          0
1 X     35  wantq = 1      2          1          0
-------------
depth-limit (-u15 steps) reached
#processes: 2
 15:    proc  1 (X) dekker.pml:38 (state 9)
 15:    proc  0 (P) dekker.pml:20 (state 13)
2 processes created
```

Figure 3: Third output

Wantp and wantq become true. Wantp is true and turn equals 1, so wantq becomes false(critical section of proctype X). The else on line 9 has no effect because turn equals 1, not 2. Turn becomes 2 and wantp becomes false (lines 20 and 21, proctype P). Wantp and wantq become true.

```
  0:    proc  - (:root:) creates proc  0 (P)
  0:    proc  - (:root:) creates proc  1 (X)
0 P     7   wantp = 1
Process Statement          wantp
1 X     27  wantq = 1      1
Process Statement          wantp      wantq
1 X     29  wantp          1          1
1 X     31  turn==1        1          1
1 X     33  wantq = 0      1          1
0 P     9   else           1          0
0 P     20  turn = 2       1          0
-------------
depth-limit (-u10 steps) reached
#processes: 2
 10:    proc  1 (X) dekker.pml:34 (state 5)
 10:    proc  0 (P) dekker.pml:21 (state 16)
2 processes created
```

Figure 4: Fourth output
Wantp and wantq become true. Wantq is true and tur equals 1, so wantq becomes false (critical section of proctype X). The else on line 9 has no effect because turn equals 1. Turn becomes 2(proctype P).

```
  0:    proc  - (:root:) creates proc  0 (P)
  0:    proc  - (:root:) creates proc  1 (X)
0 P     7   wantp = 1
Process Statement          wantp
0 P     9   else           1
1 X     27  wantq = 1      1
Process Statement          wantp      wantq
0 P     20  turn = 2       1          1
Process Statement          turn       wantp      wantq
0 P     21  wantp = 0      2          1          1
1 X     29  else           2          0          1
0 P     7   wantp = 1      2          0          1
1 X     40  turn = 1       2          1          1
1 X     41  wantq = 0      1          1          1
0 P     9   else           1          1          0
0 P     20  turn = 2       1          1          0
1 X     27  wantq = 1      2          1          0
-------------
depth-limit (-u20 steps) reached
#processes: 2
 20:    proc  1 (X) dekker.pml:40 (state 13)
 20:    proc  0 (P) dekker.pml:21 (state 16)
2 processes created
```

Figure 5: Fifth output
Wantp becomes true. Theelse on line 9 has no effect because turn equals 1. Wantp becomes true; turn becomes true and wantp becomes false. The else on line 29 has no effect because wantp is false. Wantp becomes true. Turn becomes 1 and wantq becomes false. the else on line 9 has no effect because wantq is false. Turn becomes 2 and wantq becomes true.

4 Conclusions

The critical section has to finish it's execution, before anything else is done. Another thread waits to enter in execution until the first critical section is finished.
This problem is a good example for critical section, because you can't know which proctype is going to enter the critical section first. Using jspin and Promela we can see better how interthreading works and which proctype enters the critical section, when and why.