PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
ESCUELA DE POSGRADO
MAESTRÍA EN INFORMÁTICA

INF646 MÉTODOS FORMALES
Examen 2
2018 – 2

Prepare un directorio de trabajo con el nombre <*su-código-de-8-dígitos*>, por ejemplo, `20174160`.
Este directorio es para desarrollar los programas de las preguntas del examen. Los nombres de los programas se indican en las preguntas.
Las respuestas a las preguntas y su comentarios usted puede preparar en el archivo <*su-código-de-8-dígitos*>**.txt**.
Al final del examen, comprime todo el directorio de trabajo al archivo <*su-código-de-8-dígitos*>**.zip** y colóquelo en la carpeta **Documentos del curso/Examen 2/Buzón/** en el Campus Virtual.

A esta hoja están acompañando los 5 archivos: "**Readers-Writers v5.pdf**", **rdr_wrt_msg_v5.pml**, **rdr_wrt_msg_v9.pml**, **sched1_v6_b.pml** y **sched2_v6.pml**. Cópialos a su directorio de trabajo.

Pregunta 1. (**4 puntos – 36 min.**) En la versión 5 (incorrecta) del modelo para lectores y escritores la salida es la siguiente:

```
$ spin rdr_wrt_msg_v5.pml | expand
          t=0: request from Reader 0
          t=0: OK to Reader 0
              t=0: reader 0
          t=1: request from Reader 1
          t=1: OK to Reader 1
                  t=1: reader 1
          t=2: finished Reader 0
          t=2: request from Writer 2
      timeout
                                        t: 2 -> 3
          t=3: finished Reader 1
          t=3: OK to Writer 2
                       t=3: writer 2
      timeout
                                        t: 4 -> 5
      timeout
                                        t: 5 -> 6
      timeout
                                        t: 6 -> 7
          t=7: finished Writer 2
          t=7: request from Writer 5
          t=7: OK to Writer 5
                                  t=7: writer 5
      timeout
                                        t: 8 -> 9
      timeout
                                        t: 9 -> 10
          t=10: finished Writer 5
          t=10: request from Reader 3
          t=10: OK to Reader 3
                           t=10: reader 3
          t=11: request from Reader 4
```

```
            t=11: OK to Reader 4
                                t=11: reader 4
            t=12: finished Reader 3
            t=12: finished Reader 4
            t=12: request from Reader 6
            t=12: OK to Reader 6
                                        t=12: reader 6
            t=13: finished Reader 6
      timeout
                                             t: 13 -> 14
      timeout
                                             t: 14 -> 15
      timeout
                                             t: 15 -> 16
      timeout
                                             t: 16 -> 17
      timeout
                                             t: 17 -> 18
      timeout
                                             t: 18 -> 19
      timeout
                                             t: 19 -> 20
      timeout
      timeout
#processes: 2
                queue 2 (readrequest):
                queue 1 (writerequest):
                queue 3 (finished):
                queue 4 (mbox[0]):
                queue 5 (mbox[1]):
                queue 6 (mbox[2]):
                queue 7 (mbox[3]):
                queue 8 (mbox[4]):
                queue 9 (mbox[5]):
                queue 10 (mbox[6]):
                start[0] = 0
                start[1] = 1
                start[2] = 2
                start[3] = 3
                start[4] = 4
                start[5] = 5
                start[6] = 6
                nr = 0
                nw = 0
                t = 20
305:    proc  1 (Controller:1) rdr_wrt_msg_v5.pml:52 (state 37) <valid end state>
305:    proc  0 (:init::1) rdr_wrt_msg_v5.pml:128 (state 20) <valid end state>
10 processes created
```

Lo que significa que el proceso `Idle` sigue trabajando a pesar que los procesos de lectores y escritores ya se acabaron, lo debería hacer también el proceso `Idle`, pero el modelo no termina. Explique cómo se logra la siguiente salida cuando el modelo termina al terminar los procesos de lectores y escritores:

```
$ spin rdr_wrt_msg_v5a.pml | expand
          t=0: request from Reader 0
          t=0: OK to Reader 0
                  t=0: reader 0
          t=1: request from Reader 1
          t=1: OK to Reader 1
                       t=1: reader 1
          t=2: finished Reader 0
```

```
            t=2: request from Writer 2
      timeout
              t: 2 -> 3
            t=3: finished Reader 1
            t=3: OK to Writer 2
                          t=3: writer 2
      timeout
              t: 4 -> 5
      timeout
              t: 5 -> 6
      timeout
              t: 6 -> 7
            t=7: finished Writer 2
            t=7: request from Writer 5
            t=7: OK to Writer 5
                                      t=7: writer 5
      timeout
              t: 8 -> 9
      timeout
              t: 9 -> 10
            t=10: finished Writer 5
            t=10: request from Reader 3
            t=10: OK to Reader 3
                               t=10: reader 3
            t=11: finished Reader 3
            t=11: request from Reader 4
            t=11: OK to Reader 4
                                  t=11: reader 4
            t=12: request from Reader 6
            t=12: OK to Reader 6
                                         t=12: reader 6
            t=13: finished Reader 4
            t=13: finished Reader 6
      timeout
      timeout
#processes: 2
                queue 1 (readrequest):
                queue 2 (writerequest):
                queue 10 (finished):
                queue 3 (mbox[0]):
                queue 4 (mbox[1]):
                queue 5 (mbox[2]):
                queue 6 (mbox[3]):
                queue 7 (mbox[4]):
                queue 8 (mbox[5]):
                queue 9 (mbox[6]):
                start[0] = 0
                start[1] = 1
                start[2] = 2
                start[3] = 3
                start[4] = 4
                start[5] = 5
                start[6] = 6
                nr = 0
                nw = 0
                t = 13
277:    proc  1 (Controller:1) rdr_wrt_msg_v5a.pml:52 (state 37) <valid end state>
277:    proc  0 (:init::1) rdr_wrt_msg_v5a.pml:128 (state 20) <valid end state>
10 processes created
```

Pregunta 2 (8 puntos – 1 hora 12 min.) La versión **`rdr_wrt_msg_v9.pml`** produce la siguiente salida en una ejecución aleatoria (no cualquiera):

```
$ spin rdr_wrt_msg_v9.pml | expand
                t=0: reader 0 send a request
          t=0: request from Reader 0
          t=0: OK to Reader 0
                  t=0: reader 0 received ok waiting 0
                      t=1: reader 1 send a request
          t=1: request from Reader 1
          t=1: OK to Reader 1
                      t=1: reader 1 received ok waiting 0
                          t=2: writer 2 send a request
          t=2: request from Writer 2
      timeout
              t: 2 -> 3
          t=3: finished Reader 0
                               t=3: reader 3 send a request
      timeout
              t: 3 -> 4
          t=4: finished Reader 1
          t=4: OK to Writer 2
                          t=4: writer 2 received ok waiting 2
                                  t=5: reader 4 send a request
                                      t=5: writer 5 send a request
      timeout
              t: 5 -> 6
                                           t=6: reader 6 send a request
      timeout
              t: 6 -> 7
      timeout
              t: 7 -> 8
      timeout
              t: 8 -> 9
          t=9: finished Writer 2
          t=9: request from Writer 5
          t=9: OK to Writer 5
                                      t=9: writer 5 received ok waiting 4
      timeout
              t: 10 -> 11
      timeout
              t: 11 -> 12
      timeout
              t: 12 -> 13
      timeout
              t: 13 -> 14
          t=14: finished Writer 5
          t=14: request from Reader 3
          t=14: OK to Reader 3
                               t=14: reader 3 received ok waiting 11
          t=14: request from Reader 4
          t=14: OK to Reader 4
                                    t=14: reader 4 received ok waiting 10
          t=16: request from Reader 6
          t=16: OK to Reader 6
                                           t=16: reader 6 received ok waiting 10
          t=17: finished Reader 3
          t=17: finished Reader 4
      timeout
              t: 17 -> 18
      timeout
              t: 18 -> 19
```

```
          t=19: finished Reader 6
      timeout
      timeout
#processes: 2
              queue 1 (readrequest):
              queue 2 (writerequest):
              queue 3 (finished):
              queue 4 (mbox[0]):
              queue 5 (mbox[1]):
              queue 6 (mbox[2]):
              queue 7 (mbox[3]):
              queue 8 (mbox[4]):
              queue 9 (mbox[5]):
              queue 10 (mbox[6]):
              queue 11 (mbox[7]):
              count = 100
              start[0] = 0
              start[1] = 1
              start[2] = 2
              start[3] = 3
              start[4] = 4
              start[5] = 5
              start[6] = 6
              waiting[0] = 0
              waiting[1] = 0
              waiting[2] = 2
              waiting[3] = 11
              waiting[4] = 10
              waiting[5] = 4
              waiting[6] = 10
              nr = 0
              nw = 0
              t = 19
330:   proc  1 (Controller:1) rdr_wrt_msg_v9.pml:63 (state 20) <valid end state>
330:   proc  0 (:init::1) rdr_wrt_msg_v9.pml:137 (state 20) <valid end state>
10 processes created
```

El primer error se puede observar en la línea

```
                                t=5: reader 4 send a request
```

porque esta solicitud hubiera que producirse en el tiempo 4.

¿Cómo se corrige este error obteniendo siempre la salida como la siguiente?

```
$ spin rdr_wrt_msg_v9a.pml | expand
                    t=0: reader 0 send a request
          t=0: processing request from Reader 0
          t=0: OK to Reader 0
                    t=0: reader 0 received ok waiting 0
      timeout
              t: 0 -> 1
                        t=1: reader 1 send a request
          t=1: processing request from Reader 1
          t=1: OK to Reader 1
                        t=1: reader 1 received ok waiting 0
      timeout
              t: 1 -> 2
                             t=2: writer 2 send a request
          t=2: processing request from Writer 2
      timeout
```

```
            t: 2 -> 3
        t=3: finished Reader 0
                              t=3: reader 3 send a request
    timeout
            t: 3 -> 4
        t=4: finished Reader 1
                                 t=4: reader 4 send a request
        t=4: OK to Writer 2
                           t=4: writer 2 received ok waiting 2
    timeout
            t: 4 -> 5
                                       t=5: writer 5 send a request
    timeout
            t: 5 -> 6
                                           t=6: reader 6 send a request
    timeout
            t: 6 -> 7
    timeout
            t: 7 -> 8
    timeout
            t: 8 -> 9
        t=9: finished Writer 2
        t=9: processing request from Writer 5
        t=9: OK to Writer 5
                                       t=9: writer 5 received ok waiting 4
    timeout
            t: 9 -> 10
    timeout
            t: 10 -> 11
    timeout
            t: 11 -> 12
    timeout
            t: 12 -> 13
    timeout
            t: 13 -> 14
        t=14: finished Writer 5
        t=14: processing request from Reader 3
        t=14: OK to Reader 3
                               t=14: reader 3 received ok waiting 11
        t=14: processing request from Reader 4
        t=14: OK to Reader 4
                                  t=14: reader 4 received ok waiting 10
        t=14: processing request from Reader 6
        t=14: OK to Reader 6
                                           t=14: reader 6 received ok waiting 8
    timeout
            t: 14 -> 15
    timeout
            t: 15 -> 16
    timeout
            t: 16 -> 17
        t=17: finished Reader 3
        t=17: finished Reader 6
        t=17: finished Reader 4
    timeout
    timeout
#processes: 2
              queue 3 (readrequest):
              queue 1 (writerequest):
              queue 2 (finished):
              queue 4 (mbox[0]):
              queue 5 (mbox[1]):
              queue 6 (mbox[2]):
```

```
                queue 7 (mbox[3]):
                queue 8 (mbox[4]):
                queue 9 (mbox[5]):
                queue 10 (mbox[6]):
                queue 11 (mbox[7]):
                count = 100
                start[0] = 0
                start[1] = 1
                start[2] = 2
                start[3] = 3
                start[4] = 4
                start[5] = 5
                start[6] = 6
                waiting[0] = 0
                waiting[1] = 0
                waiting[2] = 2
                waiting[3] = 11
                waiting[4] = 10
                waiting[5] = 4
                waiting[6] = 8
                nr = 0
                nw = 0
                t = 17
343:    proc  1 (Controller:1) rdr_wrt_msg_v9a.pml:74 (state 20) <valid end state>
343:    proc  0 (:init::1) rdr_wrt_msg_v9a.pml:148 (state 20) <valid end state>
10 processes created
```

Pregunta 3 (4 puntos – 36 min.) Hasta ahora los asertos verificaban solamente la exclusión mutua del algoritmo. En la versión **`rdr_wrt_msg_v10.pml`** se introducen los asertos para verificar el algoritmo del proceso `Controller`:

```
$ cat -n rdr_wrt_msg_v10.pml | expand
...
    69  proctype Controller() {
    70      byte r,w    // process id
    71
    72      do
    73      ::  count > 0 ->    // readers are welcome
    74          assert(... sobre nw)
    75  end:    if
    76          ::  nempty(finished) ->
    77                  atomic {
    78                      finished ? r
    79                      printf("t=%d: finished Reader %d\n",t,r)
    80                  }
    81                  count++
    82                  assert(... relación entre nr y count)
    83          ::  empty(finished) && nempty(writerequest) ->
    84                  atomic {
    85                  writerequest ? w
    86                      printf("t=%d: processing request from Writer %d\n",t,w)
    87                  }
    88                  count = count - MAXRDRS    // no more readers
    89                  assert(... relación entre nr y count)
    90          ::  empty(finished) && empty(writerequest) && nempty(readrequest) ->
    91                  atomic {
    92                      readrequest ? r
    93                      printf("t=%d: processing request from Reader %d\n",t,r)
    94                  }
```

```
 95                  count--
 96                  assert(... sobre count)
 97                  atomic {
 98                      mbox[r] ! true  // send ok to reader
 99                      printf("t=%d: OK to Reader %d\n",t,r)
100                  }
101          fi
102      ::  count == 0 ->   // there aren't readers, writer may go
103              assert(...)
104              atomic {
105                  mbox[w] ! true  // send ok to writer
106                  printf("t=%d: OK to Writer %d\n",t,w)
107              }
108              atomic {
109                  finished ? w    // wait writer finishing
110                  printf("t=%d: finished Writer %d\n",t,w)
111              }
112              count = MAXRDRS     // initial state
113      ::  count < 0 ->    // writer is waiting because readers access
114              assert(... relación entre nr y count)
115              atomic {
116                  finished ? r
117                  printf("t=%d: finished Reader %d\n",t,r)
118              }
119              count++
120      od
121  }
```

Complete el código de **`rdr_wrt_msg_v10.pml`**.

Pregunta 4. (**4 puntos – 36 min.**) Considere los modelos de los siguientes programas (**`sched1_v6_b.pml`** y **`sched2_v6.pml`**), sus simulaciones y explique los resultados que suceden durante la simulación:

```
$ cat -n sched1_v6_b.pml | expand
     1  /* Copyright 2007 by Moti Ben-Ari under the GNU GPL; see readme.txt */
     2  /* PSMC, pp.175-177
     3     vk, 2015
     4  */
     5
     6  #define N 2             /* number of processes */
     7  byte clock = 0          /* models time */
     8  bool done[N] = false    /* done before the deadline */
     9
    10  proctype T(byte ID; byte period; byte exec) {
    11      byte next = 0       /* next time to execute */
    12      do
    13      ::  d_step {
    14              clock >= next ->    /* is it time to execute? */
    15                  printf("Task %d: executed from %d ", ID, clock)
    16                  clock = clock + exec    /* executed */
    17                  printf("to %d\n", clock)
    18                  done[ID] = true
    19                  next = next + period    /* next time to execute */
    20          }
    21      od
    22  }
    23
    24  proctype Watchdog(byte ID; byte period) {       /* for every task */
```

```
25      byte deadline = period
26      do
27      ::  d_step {
28              clock >= deadline ->
29                  assert done[ID]
30                  deadline = deadline + period
31                  done[ID] = false
32          }
33      od
34  }
35
36  proctype Idle() {
37      do
38      ::  d_step {
39              timeout -> {
40                  clock++
41                  printf("Idle, clock ticking: %d\n", clock)
42              }
43      }
44      od
45  }
46
47  init {
48      d_step {
49          run Idle()
50          run T(0, 2, 1)      /* Task ID, period, execution time */
51          run Watchdog(0, 2) priority 3  /* Task ID, task deadline */
52          run T(1, 5, 2)
53          run Watchdog(1, 5) priority 3
54          printf("init priority      = %d\n",get_priority(0))
55          printf("Idle priority      = %d\n",get_priority(1))
56          printf("Task0 priority     = %d\n",get_priority(2))
57          printf("Watchdog0 priority = %d\n",get_priority(3))
58          printf("Task1 priority     = %d\n",get_priority(4))
59          printf("Watchdog1 priority = %d\n",get_priority(5))
60      }
61  }
```

```
$ spin -T -h -n1449195551 sched1_v6_b.pml
init priority      = 1
Idle priority      = 1
Task0 priority     = 1
Watchdog0 priority = 3
Task1 priority     = 1
Watchdog1 priority = 3
Task 0: executed from 0 to 1
Task 1: executed from 1 to 3
Task 0: executed from 3 to 4
Task 0: executed from 4 to 5
Task 1: executed from 5 to 7
Task 0: executed from 7 to 8
Task 0: executed from 8 to 9
timeout
Idle, clock ticking: 10
Task 0: executed from 10 to 11
Task 1: executed from 11 to 13
Task 0: executed from 13 to 14
Task 0: executed from 14 to 15
Task 1: executed from 15 to 17
Task 0: executed from 17 to 18
Task 0: executed from 18 to 19
timeout
Idle, clock ticking: 20
```

```
Task 0: executed from 20 to 21
Task 1: executed from 21 to 23
Task 0: executed from 23 to 24
Task 0: executed from 24 to 25
Task 1: executed from 25 to 27
Task 0: executed from 27 to 28
Task 0: executed from 28 to 29
timeout
Idle, clock ticking: 30
Task 0: executed from 30 to 31
Task 1: executed from 31 to 33
Task 0: executed from 33 to 34
Task 0: executed from 34 to 35
Task 1: executed from 35 to 37
Task 0: executed from 37 to 38
Task 0: executed from 38 to 39
timeout
Idle, clock ticking: 40
Task 0: executed from 40 to 41
Task 1: executed from 41 to 43
Task 0: executed from 43 to 44
Task 0: executed from 44 to 45
Task 1: executed from 45 to 47
Task 0: executed from 47 to 48
Task 0: executed from 48 to 49
timeout
Idle, clock ticking: 50
Task 0: executed from 50 to 51
Task 1: executed from 51 to 53
Task 0: executed from 53 to 54
Task 0: executed from 54 to 55
Task 1: executed from 55 to 57
Task 0: executed from 57 to 58
Task 0: executed from 58 to 59
timeout
Idle, clock ticking: 60
Task 0: executed from 60 to 61
Task 1: executed from 61 to 63
Task 0: executed from 63 to 64
Task 0: executed from 64 to 65
Task 1: executed from 65 to 67
Task 0: executed from 67 to 68
Task 0: executed from 68 to 69
timeout
Idle, clock ticking: 70
Task 1: executed from 70 to 72
spin: sched1_v6_b.pml:29, Error: assertion violated
spin: text of failed assertion: assert(done[ID])
#processes: 6
        clock = 72
        done[0] = 0
        done[1] = 1
530:    proc  5 (Watchdog:3) sched1_v6_b.pml:26 (state 6)
530:    proc  4 (T:1) sched1_v6_b.pml:12 (state 8)
530:    proc  3 (Watchdog:3) sched1_v6_b.pml:29 (state 2)
530:    proc  2 (T:1) sched1_v6_b.pml:12 (state 8)
530:    proc  1 (Idle:1) sched1_v6_b.pml:37 (state 6)
530:    proc  0 (:init::1) sched1_v6_b.pml:61 (state 13) <valid end state>
6 processes created
seed used: 1449195551
```

```
$ cat -n sched2_v6.pml | expand
     1  /* Copyright 2007 by Moti Ben-Ari under the GNU GPL; see readme.txt */
     2  /* PSMC, pp.177-178
     3     vk, 2015
     4  */
     5
     6  #define N 2
     7  byte clock = 0
     8
     9  proctype T(byte ID; byte period; byte exec) {
    10      byte next = 0
    11      byte deadline = period
    12      bool done = false
    13      do
    14      ::  atomic {
    15              (clock >= next) && (clock < deadline) ->
    16                  printf("Task %d: executed from %d ", ID, clock)
    17                  clock = clock + exec    /* executed */
    18                  printf("to %d\n", clock)
    19                  next = next + period
    20                  done = true
    21          }
    22      ::  atomic {
    23              clock >= deadline ->
    24                  assert done
    25                  deadline = deadline + period
    26                  done = false
    27          }
    28      od
    29  }
    30
    31  proctype Idle() {
    32      do
    33      ::  atomic {
    34              timeout -> {
    35              clock++
    36              printf("Idle, clock ticking: %d\n", clock)
    37              }
    38          }
    39      od
    40  }
    41
    42  init {
    43      atomic {
    44          run Idle()
    45          run T(0, 2, 1)
    46          run T(1, 5, 2)
    47      }
    48  }
```

```
$ spin -T -h -n1449243584 sched2_v6.pml
Task 0: executed from 0 to 1
Task 1: executed from 1 to 3
Task 0: executed from 3 to 4
Task 0: executed from 4 to 5
Task 1: executed from 5 to 7
Task 0: executed from 7 to 8
Task 0: executed from 8 to 9
timeout
Idle, clock ticking: 10
Task 0: executed from 10 to 11
```

```
Task 1: executed from 11 to 13
Task 0: executed from 13 to 14
Task 0: executed from 14 to 15
Task 1: executed from 15 to 17
Task 0: executed from 17 to 18
Task 0: executed from 18 to 19
timeout
Idle, clock ticking: 20
Task 0: executed from 20 to 21
Task 1: executed from 21 to 23
Task 0: executed from 23 to 24
Task 0: executed from 24 to 25
Task 1: executed from 25 to 27
Task 0: executed from 27 to 28
Task 0: executed from 28 to 29
timeout
Idle, clock ticking: 30
Task 1: executed from 30 to 32
spin: sched2_v6.pml:24, Error: assertion violated
spin: text of failed assertion: assert(done)
#processes: 4
                clock = 32
275:    proc  3 (T:1) sched2_v6.pml:29 (state 14)
275:    proc  2 (T:1) sched2_v6.pml:24 (state 9)
275:    proc  1 (Idle:1) sched2_v6.pml:32 (state 6)
275:    proc  0 (:init::1) sched2_v6.pml:48 (state 5) <valid end state>
4 processes created
seed used: 1449243584
```

Profesor: V. Khlebnikov
Pando, 7 de diciembre de 2018