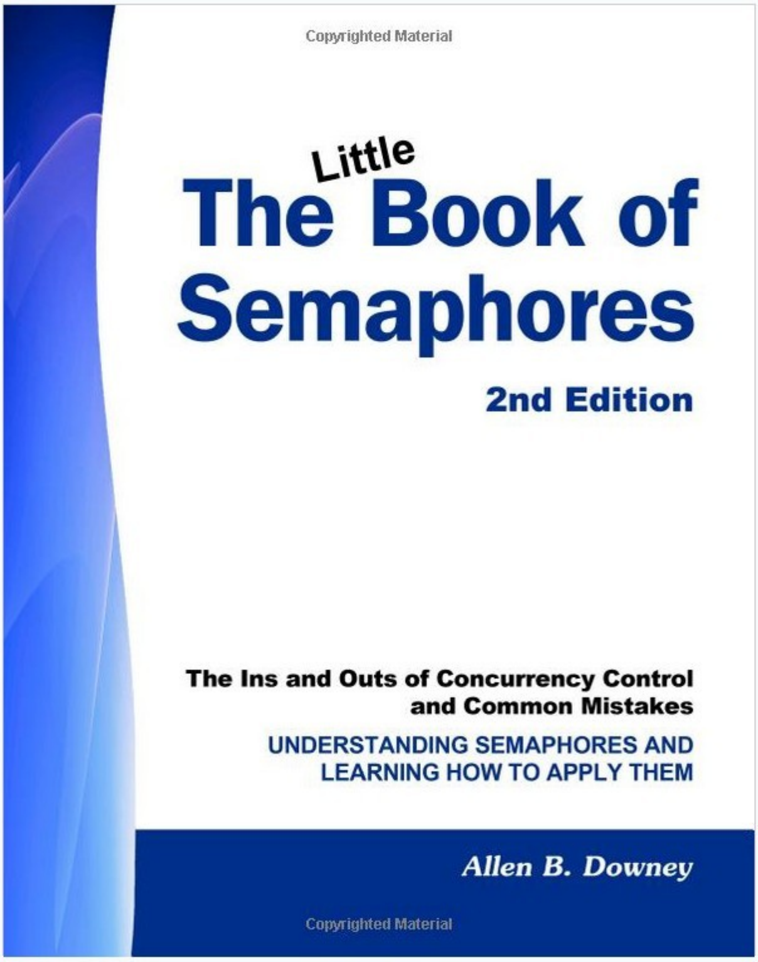


Copyrighted Material

The Little Book of Semaphores

2nd Edition

The Ins and Outs of Concurrency Control and Common Mistakes

UNDERSTANDING SEMAPHORES AND LEARNING HOW TO APPLY THEM

Allen B. Downey

Copyrighted Material

Allen B. Downey

The Little Book of Semaphores

Version 2.2.1

http://www.greenteapress.com/semaphores/LittleBookOfSemaphores.pdf

TLBOS, Chapter 3

Basic synchronization patterns

3.1 Signaling

Signaling makes it possible to guarantee that a section of code in one thread will run before a section of code in another thread; in other words, it solves the **serialization problem**.

The semaphore in the next program guarantee that the process **A** has completed the assignment to the variable **x** before the process **B** begins its assignment to the same variable.

3.1 Signaling (3.1.signaling.pml)

```
#define wait(sem)   atomic { sem > 0; sem-- }
#define signal(sem) sem++

byte sem = 0
byte x = 0

proctype A() {
  x = 1
  signal(sem)
}

proctype B() {
  wait(sem)
  x = 2
}

init {
  atomic { run A(); run B() }
  _nr_pr == 1
  assert( x == 2 )
}
```

3.1 Signaling (`3.1.signaling.pml`)

```
$ spin -run 3.1.signaling.pml


(Spin Version 6.4.8 -- 2 March 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 28 byte, depth reached 11, errors: 0
...
```

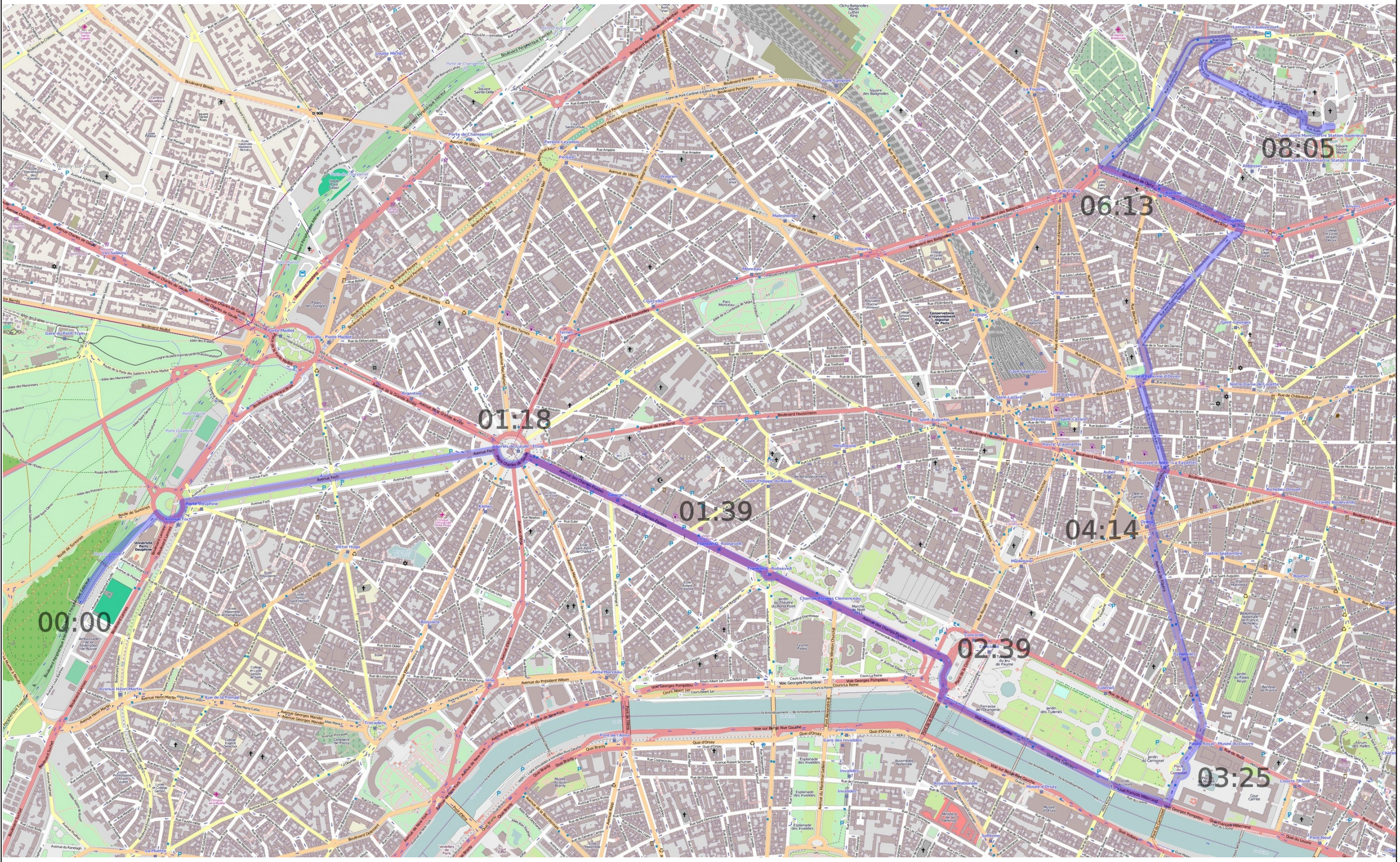
TLBOS, Chapter 3

Basic synchronization patterns

3.3 Rendezvous

The idea is that two threads rendezvous at a point of execution, and neither is allowed to proceed until both have arrived.

Claude Lelouch, 1976, 8 min 38 seconds



3.3 Rendezvous (3.3.0.rendezvous.pml)

```
$ cat -n 3.3.0.rendezvous.pml
     1 /*  The Little Book of Semaphores (2.2.1)
     2     by A. Downey
     3
     4     Chapter 3. Basic synchronization patterns
     5
     6     3.1 Signaling
     7     3.3 Rendezvous
     8                    Thread A                Thread 2
     9                    1  statement a1         1  statement b1
    10                    2  statement a2         2  statement b2
    11
    12     We want to guarantee that a1 happens before b2 and b1 happens
before a2:
    13         a1,b1,b2,a2;   a1,b1,a2,b2;   b1,a1,a2,b2;   b1,a1,b2,a2
    14     prohibiting
    15         b1,b2,a1,a2;   a1,a2,b1,b2
    16
    17     3.3.0.rendezvous.pml: all 6 possible sequences
    18 */
    19
...
```

3.3 Rendezvous (3.3.0.rendezvous.pml)

```
...
    20 int x = 0
    21
    22 proctype A() {
    23   x = 10*x + 1
    24   x = 10*x + 2
    25 }
    26
    27 proctype B() {
    28   x = 10*x + 3
    29   x = 10*x + 4
    30 }
    31
    32 init {
    33   atomic { run A(); run B() }
    34   _nr_pr == 1
    35   printf("x = %d\n", x)
    36   assert(x==1234 || x==1324 || x==1342 || x==3412 || x==3142 ||
x==3124)
    37 /* must be prohibited: 3412 and 1234 */
    38 }
```

3.3 Rendezvous (3.3.0.rendezvous.pml)

```
$ spin 3.3.0.rendezvous.pml
      x = 1342
3 processes created

$ spin 3.3.0.rendezvous.pml
      x = 1234
3 processes created

$ spin 3.3.0.rendezvous.pml
      x = 1234
3 processes created

$ spin 3.3.0.rendezvous.pml
      x = 3142
3 processes created

$ spin 3.3.0.rendezvous.pml
      x = 3412
3 processes created

...
```

3.3 Rendezvous (3.3.0.rendezvous.pml)

```
$ spin -run 3.3.0.rendezvous.pml

(Spin Version 6.4.8 -- 2 March 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 36 byte, depth reached 12, errors: 0
...
```

Only these 6 sequences are possible, but two of them are prohibited!

3.3.2 Rendezvous (efficient) solution (`3.3.2a.rendezvous.pml`)

```
$ cat -n 3.3.2a.rendezvous.pml
     1 /*  The Little Book of Semaphores (2.2.1)
     2     by A. Downey
     3
     4     Chapter 3. Basic synchronization patterns
     5
     6     3.1 Signaling
     7     3.3 Rendezvous
     8                    Thread A                Thread 2
     9                    1  statement a1         1  statement b1
    10                    2  statement a2         2  statement b2
    11
    12     We want to guarantee that a1 happens before b2 and b1 happens
before a2:
    13         a1,b1,b2,a2; a1,b1,a2,b2; b1,a1,a2,b2; b1,a1,b2,a2
    14     prohibiting
    15         b1,b2,a1,a2; a1,a2,b1,b2
    16
    17     3.3.2a Rendezvous solution (efficient)
    18 */
    19
...
```

3.3.2 Rendezvous (efficient) solution (`3.3.2a.rendezvous.pml`)

```
...
    20 #define wait(sem)   atomic { sem > 0; sem-- }
    21 #define signal(sem) sem++
    22
    23 byte aArrived = 0, bArrived = 0
    24 int x = 0
    25
    26 proctype A() {
    27   x = 10*x + 1
    28   signal(aArrived)   # a) "llegaré en 10 minutos"
    29   wait(bArrived)     # b) llega en 8, debe esperar (context switch)
    30   x = 10*x + 2
    31 }
    32
    33 proctype B() {
    34   x = 10*x + 3
    35   signal(bArrived)   # c) "ya llegué"
    36   wait(aArrived)     # d) puede seguir sin cambio del contexto (1342)
    37   x = 10*x + 4
    38 }
    39
...
```

3.3.2 Rendezvous (efficient) solution (`3.3.2a.rendezvous.pml`)

```
...
init {
  atomic { run A(); run B() }
  _nr_pr == 1
  assert(x!=1234 && x!=3412)
}
```

3.3.2 Rendezvous (efficient) solution (`3.3.2a.rendezvous.pml`)

```
$ spin -run 3.3.2a.rendezvous.pml

(Spin Version 6.4.8 -- 2 March 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 36 byte, depth reached 15, errors: 0
...
```

3.3.2 Rendezvous (less efficient) solution (3.3.2b.rendezvous.pml)

```
$ cat -n 3.3.2b.rendezvous.pml
     1	/*  The Little Book of Semaphores (2.2.1)
     2	    by A. Downey
     3	
     4	    Chapter 3. Basic synchronization patterns
     5	
     6	    3.1 Signaling
     7	    3.3 Rendezvous
     8	                   Thread A                Thread 2
     9	                   1  statement a1         1  statement b1
    10	                   2  statement a2         2  statement b2
    11	
    12	    We want to guarantee that a1 happens before b2 and b1 happens
before a2:
    13	        a1,b1,b2,a2; a1,b1,a2,b2; b1,a1,a2,b2; b1,a1,b2,a2
    14	    prohibiting
    15	        b1,b2,a1,a2; a1,a2,b1,b2
    16	
    17	    3.3.2b Rendezvous solution (less efficient)
    18	*/
    19	
...
```

3.3.2 Rendezvous (less efficient) solution (3.2.2b.rendezvous.pml)

```
...
    20	#define wait(sem)   atomic { sem > 0; sem-- }
    21	#define signal(sem) sem++
    22	
    23	byte aArrived = 0, bArrived = 0
    24	int x = 0
    25	
    26	proctype A() {
    27	  x = 10*x + 1
    28	  wait(bArrived)      # a) "¿cuándo llegaras?" (context switch)
    29	  signal(aArrived)    # d) "pardon, ya estoy" (1324 o 1342)
    30	  x = 10*x + 2
    31	}
    32	
    33	proctype B() {
    34	  x = 10*x + 3
    35	  signal(bArrived)    # b) "ya llegué"
    36	  wait(aArrived)      # c) "pero tú no estás" (context switch extra)
    37	  x = 10*x + 4
    38	}
    39	
...
```

3.3.2 Rendezvous (less efficient) solution (3.3.2b.rendezvous.pml)

```
...
    init {
      atomic { run A(); run B() }
      _nr_pr == 1
      assert(x!=1234 && x!=3412)
    }

```

3.3.2 Rendezvous (less efficient) solution (3.3.2b.rendezvous.pml)

```
$ spin -a 3.3.2b.rendezvous.pml

(Spin Version 6.4.8 -- 2 March 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 36 byte, depth reached 15, errors: 0
...
```

3.3.3 Rendezvous Deadlock #1 (3.3.3.rendezvous.pml)

```
$ cat -n 3.2.3.rendezvous.pml
     1	/*  The Little Book of Semaphores (2.1.5)
     2	    by A. Downey
     3
     4	    Chapter 3. Basic synchronization patterns
     5
     6	    3.1 Signaling
     7	    3.2 Rendezvous
     8	                   Thread A                Thread 2
     9	                   1  statement a1         1  statement b1
    10	                   2  statement a2         2  statement b2
    11
    12	    We want to guarantee that a1 happens before b2 and b1 happens
before a2:
    13	        a1,b1,b2,a2; a1,b1,a2,b2; b1,a1,a2,b2; b1,a1,b2,a2
    14	    prohibiting
    15	        b1,b2,a1,a2; a1,a2,b1,b2
    16
    17	    3.2.3 Deadlock #1
    18	*/
    19
...
```

3.3.3 Rendezvous Deadlock #1 (3.3.3.rendezvous.pml)

```
...
    20	#define wait(sem)   atomic { sem > 0; sem-- }
    21	#define signal(sem) sem++
    22
    23	byte aArrived = 0, bArrived = 0;
    24	int x = 0
    25
    26	proctype A() {
    27	  x = 10*x + 1
    28	  wait(bArrived)
    29	  signal(aArrived)
    30	  x = 10*x + 2
    31	}
    32
    33	proctype B() {
    34	  x = 10*x + 3
    35	  wait(aArrived)
    36	  signal(bArrived)
    37	  x = 10*x + 4
    38	}
    39
...
```

3.3.3 Rendezvous Deadlock #1 (`3.3.3.rendezvous.pml`)

```
...
    init {
      atomic { run A(); run B() }
      _nr_pr == 1
      assert(x!=1234 && x!=3412)
    }
```

3.3.3 Rendezvous Deadlock #1 (`3.3.3.rendezvous.pml`)

```
$ spin -run 3.3.3.rendezvous.pml
pan:1: invalid end state (at depth 3)
pan: wrote 3.3.3.rendezvous.pml.trail

(Spin Version 6.4.8 -- 2 March 2018)
Warning: Search not completed
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 36 byte, depth reached 4, errors: 1
...
```

3.3.3 Rendezvous Deadlock #1 (3.3.3.rendezvous.pml)

```
$ spin 3.3.3.rendezvous.pml
      timeout
#processes: 3
                aArrived = 0
                bArrived = 0
                x = 31
  4:    proc  2 (B:1) 3.3.3.rendezvous.pml:35 (state 4)
  4:    proc  1 (A:1) 3.3.3.rendezvous.pml:28 (state 4)
  4:    proc  0 (:init::1) 3.3.3.rendezvous.pml:42 (state 4)
3 processes created
```

3.3.3 Rendezvous Deadlock #1 (3.3.3.rendezvous.pml)

```
$ spin -p 3.3.3.rendezvous.pml
  0:    proc  - (:root:) creates proc  0 (:init:)
Starting A with pid 1
  1:    proc  0 (:init::1) creates proc  1 (A)
  1:    proc  0 (:init::1) 3.3.3.rendezvous.pml:41 (state 1)    [(run A())]
Starting B with pid 2
  2:    proc  0 (:init::1) creates proc  2 (B)
  2:    proc  0 (:init::1) 3.3.3.rendezvous.pml:41 (state 2)    [(run B())]
  3:    proc  1 (A:1) 3.3.3.rendezvous.pml:27 (state 1) [x = ((10*x)+1)]
  4:    proc  2 (B:1) 3.3.3.rendezvous.pml:34 (state 1) [x = ((10*x)+3)]
      timeout
#processes: 3
                aArrived = 0
                bArrived = 0
                x = 13
  4:    proc  2 (B:1) 3.3.3.rendezvous.pml:35 (state 4)
  4:    proc  1 (A:1) 3.3.3.rendezvous.pml:28 (state 4)
  4:    proc  0 (:init::1) 3.3.3.rendezvous.pml:42 (state 4)
3 processes created
```

3.4.0 Shared variable (`3.4.0.shared_var.pml`)

```
$ cat -n 3.4.0.shared_var.pml

     1 /*  The Little Book of Semaphores (2.2.1)
     2     by A. Downey
     3
     4     Chapter 3. Basic synchronization patterns
     5
     6     3.4 Mutex
     7                    Thread A                   Thread B
     8                    1  count = count + 1       1  count = count + 1
     9
    10     3.4.0.shared_var.pml
    11 */
    12
...
```

3.4.0 Shared variable (`3.4.0.shared_var.pml`)

```
...
    13 byte count = 0
    14
    15 proctype Th(byte i) {
    16   byte temp
    17
    18   temp = count
    19   count = temp + 1
    20   printf("%c: count=%d\n",i,count)
    21 }
    22
    23 init {
    24   atomic { run Th('A'); run Th('B') }
    25   _nr_pr == 1
    26   assert(count==2)
    27 }
```

3.4.0 Shared variable (3.4.0.shared_var.pml)

```
$ spin 3.4.0.shared_var.pml
              B: count=1
          A: count=2
3 processes created
```

```
$ spin 3.4.0.shared_var.pml
              B: count=1
          A: count=1
spin: 3.4.0.shared_var.pml:26, Error: assertion violated
spin: text of failed assertion: assert((count==2))
#processes: 1
                count = 1
 10:    proc  0 (:init::1) 3.4.0.shared_var.pml:26 (state 5)
3 processes created
```

3.4.2 Mutex (3.4.2.mutex.pml)

```
$ cat -n 3.4.2.mutex.pml

    1 /*  The Little Book of Semaphores (2.2.1)
    2     by A. Downey
    3
    4     Chapter 3. Basic synchronization patterns
    5
    6     3.4 Mutex
    7                Thread A                 Thread B
    8                1  mutex.wait()          1  mutex.wait()
    9                2    # critical section  2    # critical section
   10                3    count = count + 1   3    count = count + 1
   11                4  mutex.signal()        4  mutex.signal()
   12
   13     3.4.2.mutex.pml
   14 */
   15
...
```

3.4.2 Mutex (3.4.2.mutex.pml)

```
...
#define wait(sem)   atomic { sem > 0; sem-- }
#define signal(sem) sem++

byte mutex = 1
byte count = 0

proctype Th(byte i) {
  byte temp

  wait(mutex)
    temp = count
    count = temp + 1
  signal(mutex)
}

init {
  atomic { run Th('A'); run Th('B') }
  _nr_pr == 1
  assert(count==2)
}
```

3.4.2 Mutex (3.4.2.mutex.pml)

```
$ spin -run 3.4.2.mutex.pml

(Spin Version 6.4.8 -- 2 March 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 28 byte, depth reached 15, errors: 0

...
```

3.5.1 Multiplex (3.5.1.multiplex.pml)

```
$ cat -n 3.5.1.multiplex.pml | expand

     1  /*  The Little Book of Semaphores (2.2.1)
     2      by A. Downey
     3
     4      Chapter 3. Basic synchronization patterns
     5
     6      3.4 Multiplex
     7                 Thread i
     8                 1  multiplex.wait()
     9                 2    # critical section
    10                 3  multiplex.signal()
    11
    12      3.5.1.multiplex.pml
    13  */
    14
...
```

3.5.1 Multiplex (3.5.1.multiplex.pml)

```
...
    15  #define wait(sem)   atomic { sem > 0; sem-- }
    16  #define signal(sem) sem++
    17
    18  #define LIMIT 3
    19
    20  byte multiplex=LIMIT, cs=0
    21
    22  proctype Th(byte i) {
    23    wait(multiplex)
    24      cs++   /* atomic inc by Promela */
    25      assert(cs <= LIMIT)
    26      cs--   /* atomic dec by Promela */
    27    signal(multiplex)
    28  }
    29
    30  init {
    31    byte i
    32
    33    atomic {
    34      for (i : 1 .. 9) {
    35        run Th(i)
    36      }
    37    }
    38  }
```

3.5.1 Multiplex (3.5.1.multiplex.pml)

```
$ spin -run 3.5.1.multiplex.pml | expand

(Spin Version 6.4.8 -- 2 March 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 84 byte, depth reached 86, errors: 0

...

unreached in proctype Th
        (0 of 8 states)
unreached in init
        (0 of 11 states)

pan: elapsed time 0.35 seconds
pan: rate 1372128.6 states/second
```

3.6.2 Barrier non-solution (3.6.2a.barrier_nonsol.pml)

```
$ cat -n 3.6.2a.barrier_nonsol.pml | expand

     1  /*  The Little Book of Semaphores (2.2.1)
     2      by A. Downey
     3
     4      Chapter 3. Basic synchronization patterns
     5
     6      3.6 Barrier
     7      3.6.2 Barrier non-solution
     8
     9      vk, 2017
    10  */
    11
    12  #define THREADS 10   /* value for threads number */
    13  #define N       5    /* value for barrier limit */
    14
    15  #define wait(sem)   atomic { sem > 0; sem-- }
    16  #define signal(sem) sem++
    17
...
```

3.6.2 Barrier non-solution (3.6.2a.barrier_nonsol.pml)

```
...
byte count=0, mutex=1, barrier=0

proctype Th(byte i) {
    byte temp

    do
    ::  wait(mutex)
            temp=count
            count=temp+1
        signal(mutex)
        if
        :: count == N ->
                         signal(barrier)
        :: else
        fi
        wait(barrier)
        printf("Th(%d): count = %d\n",i,count)
        break
    od
}

```

3.6.2 Barrier non-solution (3.6.2a.barrier_nonsol.pml)

```
...
init {
    byte i

    atomic {
        for (i: 1 .. THREADS) {
            run Th(i)
        }
    }
}
```

3.6.2 Barrier non-solution (3.6.2a.barrier_nonsol.pml)

```
$ spin 3.6.2a.barrier_nonsol.pml | expand

                                      Th(8): count = 5
      timeout
#processes: 11
                count = 10
                mutex = 1
                barrier = 0
109:    proc 10 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  9 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  8 (Th:1) 3.6.2a.barrier_nonsol.pml:37 (state 20) <valid end
state>
109:    proc  7 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  6 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  5 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  4 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  3 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  2 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  1 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  0 (:init::1) 3.6.2a.barrier_nonsol.pml:47 (state 11) <valid end
state>
11 processes created
```

3.6.2 Barrier non-solution (3.6.2a.barrier_nonsol.pml)

```
$ spin 3.6.2a.barrier_nonsol.pml | expand

          Th(1): count = 7
      timeout
#processes: 11
                count = 10
                mutex = 1
                barrier = 0
109:    proc 10 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  9 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  8 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  7 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  6 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  5 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  4 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  3 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  2 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
109:    proc  1 (Th:1) 3.6.2a.barrier_nonsol.pml:37 (state 20) <valid end
state>
109:    proc  0 (:init::1) 3.6.2a.barrier_nonsol.pml:47 (state 11) <valid end
state>
11 processes created
```

3.6.2 Barrier non-solution (3.6.2a.barrier_nonsol.pml)

```
$ spin 3.6.2a.barrier_nonsol.pml | expand

      timeout
#processes: 11
                count = 10
                mutex = 1
                barrier = 0
104:    proc 10 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
104:    proc  9 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
104:    proc  8 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
104:    proc  7 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
104:    proc  6 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
104:    proc  5 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
104:    proc  4 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
104:    proc  3 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
104:    proc  2 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
104:    proc  1 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
104:    proc  0 (:init::1) 3.6.2a.barrier_nonsol.pml:47 (state 11) <valid end
state>
11 processes created
```

3.6.2 Barrier non-solution (3.6.2a.barrier_nonsol.pml)

```
$ spin 3.6.2a.barrier_nonsol.pml | expand
                                      Th(8): count = 5
                          Th(5): count = 5
      timeout
#processes: 11
                count = 10
                mutex = 1
                barrier = 0
114:    proc 10 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
114:    proc  9 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
114:    proc  8 (Th:1) 3.6.2a.barrier_nonsol.pml:37 (state 20) <valid end
state>
114:    proc  7 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
114:    proc  6 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
114:    proc  5 (Th:1) 3.6.2a.barrier_nonsol.pml:37 (state 20) <valid end
state>
114:    proc  4 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
114:    proc  3 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
114:    proc  2 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
114:    proc  1 (Th:1) 3.6.2a.barrier_nonsol.pml:33 (state 14)
114:    proc  0 (:init::1) 3.6.2a.barrier_nonsol.pml:47 (state 11) <valid end
state>
11 processes created
```

3.6.2 Barrier non-solution (3.6.3.barrier_nonsol.pml)

```
$ cat -n 3.6.3.barrier_nonsol.pml | expand

     1  /*  The Little Book of Semaphores (2.2.1)
     2      by A. Downey
     3
     4      Chapter 3. Basic synchronization patterns
     5
     6      3.6 Barrier
     7      3.6.2 Barrier non-solution
     8
     9      vk, 2017
    10  */
    11
    12  #define THREADS 5    /* value for threads number */
    13  #define N       5    /* value for barrier limit */
    14
    15  #define wait(sem)   atomic { sem > 0; sem-- }
    16  #define signal(sem) sem++
    17
...
```

3.6.2 Barrier non-solution (3.6.3.barrier_nonsol.pml)

```
...
    18  byte count=0, mutex=1, barrier=0
    19  bit  bar[THREADS+1]
    20
    21  proctype Th(byte i) {
    22      byte temp
    23
    24      do
    25      ::  wait(mutex)
    26              temp=count
    27              count=temp+1
    28          signal(mutex)
    29          bar[i]=false
    30          if
    31          :: count == N ->
    32              bar[i]=true
    33              assert(!bar[1]||!bar[2]||!bar[3]||!bar[4]||!bar[5])
    34              signal(barrier)
    35          :: else
    36          fi
    37          wait(barrier)
    38          printf("Th(%d): count = %d\n",i,count)
    39          break
    40      od
    41  }
```

3.6.2 Barrier non-solution (3.6.3.barrier_nonsol.pml)

```
...

init {
    byte i

    atomic {
        for (i: 1 .. THREADS) {
            run Th(i)
        }
    }
}
```

3.6.2 Barrier non-solution (3.6.3.barrier_nonsol.pml)

```
$ spin -run -E 3.6.3.barrier_nonsol.pml | expand

pan:1: assertion violated (((( !(bar[1])|| !(bar[2]))|| !(bar[3]))|| !
(bar[4]))|| !(bar[5])) (at depth 74)
pan: wrote 3.6.3.barrier_nonsol.pml.trail

(Spin Version 6.4.8 -- 2 March 2018)
Warning: Search not completed
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by -E flag)

State-vector 64 byte, depth reached 74, errors: 1
...
```

3.6.2 Barrier non-solution (3.6.3.barrier_nonsol.pml)

```
$ spin -t -p -g -l 3.6.3.barrier_nonsol.pml | expand
using statement merging
  1:    proc  0 (:init::1) 3.6.3.barrier_nonsol.pml:47 (state 1)       [i = 1]
                :init:(0):i = 1
  2:    proc  0 (:init::1) 3.6.3.barrier_nonsol.pml:47 (state 2)       [((i<=5))]
Starting Th with pid 1
  3:    proc  0 (:init::1) 3.6.3.barrier_nonsol.pml:48 (state 3)       [(run Th(i))]
  4:    proc  0 (:init::1) 3.6.3.barrier_nonsol.pml:47 (state 4)       [i = (i+1)]
                :init:(0):i = 2
...
                :init:(0):i = 6
 17:    proc  0 (:init::1) 3.6.3.barrier_nonsol.pml:49 (state 5)       [else]
 18:    proc  0 (:init::1) 3.6.3.barrier_nonsol.pml:49 (state 6)       [goto :b1]
 19:    proc  0 (:init::1) 3.6.3.barrier_nonsol.pml:49 (state 9)       [break]

 20:    proc  5 (Th:1) 3.6.3.barrier_nonsol.pml:25 (state 1)   [((mutex>0))]
 20:    proc  5 (Th:1) 3.6.3.barrier_nonsol.pml:25 (state 2)   [mutex = (mutex-1)]
                mutex = 0
 21:    proc  5 (Th:1) 3.6.3.barrier_nonsol.pml:26 (state 4)   [temp = count]
                Th(5):temp = 0
 22:    proc  5 (Th:1) 3.6.3.barrier_nonsol.pml:27 (state 5)   [count = (temp+1)]
                count = 1
 23:    proc  5 (Th:1) 3.6.3.barrier_nonsol.pml:28 (state 6)   [mutex = (mutex+1)]
                mutex = 1
 24:    proc  5 (Th:1) 3.6.3.barrier_nonsol.pml:29 (state 7)   [bar[i] = 0]
...
```

3.6.2 Barrier non-solution (3.6.3.barrier_nonsol.pml)

```
...

 25:    proc  4 (Th:1) 3.6.3.barrier_nonsol.pml:25 (state 1)   [((mutex>0))]
 25:    proc  4 (Th:1) 3.6.3.barrier_nonsol.pml:25 (state 2)   [mutex = (mutex-1)]
                mutex = 0
 26:    proc  4 (Th:1) 3.6.3.barrier_nonsol.pml:26 (state 4)   [temp = count]
                Th(4):temp = 1
 27:    proc  4 (Th:1) 3.6.3.barrier_nonsol.pml:27 (state 5)   [count = (temp+1)]
                count = 2
 28:    proc  4 (Th:1) 3.6.3.barrier_nonsol.pml:28 (state 6)   [mutex = (mutex+1)]
                mutex = 1
 29:    proc  4 (Th:1) 3.6.3.barrier_nonsol.pml:29 (state 7)   [bar[i] = 0]


 30:    proc  3 (Th:1) 3.6.3.barrier_nonsol.pml:25 (state 1)   [((mutex>0))]
 30:    proc  3 (Th:1) 3.6.3.barrier_nonsol.pml:25 (state 2)   [mutex = (mutex-1)]
                mutex = 0
 31:    proc  3 (Th:1) 3.6.3.barrier_nonsol.pml:26 (state 4)   [temp = count]
                Th(3):temp = 2
 32:    proc  3 (Th:1) 3.6.3.barrier_nonsol.pml:27 (state 5)   [count = (temp+1)]
                count = 3
 33:    proc  3 (Th:1) 3.6.3.barrier_nonsol.pml:28 (state 6)   [mutex = (mutex+1)]
                mutex = 1
 34:    proc  3 (Th:1) 3.6.3.barrier_nonsol.pml:29 (state 7)   [bar[i] = 0]

...
```

3.6.2 Barrier non-solution (3.6.3.barrier_nonsol.pml)

```
...

 35:    proc  2 (Th:1) 3.6.3.barrier_nonsol.pml:25 (state 1)   [((mutex>0))]
 35:    proc  2 (Th:1) 3.6.3.barrier_nonsol.pml:25 (state 2)   [mutex = (mutex-1)]
                mutex = 0
 36:    proc  2 (Th:1) 3.6.3.barrier_nonsol.pml:26 (state 4)   [temp = count]
                Th(2):temp = 3
 37:    proc  2 (Th:1) 3.6.3.barrier_nonsol.pml:27 (state 5)   [count = (temp+1)]
                count = 4
 38:    proc  2 (Th:1) 3.6.3.barrier_nonsol.pml:28 (state 6)   [mutex = (mutex+1)]
                mutex = 1
 39:    proc  2 (Th:1) 3.6.3.barrier_nonsol.pml:29 (state 7)   [bar[i] = 0]


 40:    proc  1 (Th:1) 3.6.3.barrier_nonsol.pml:25 (state 1)   [((mutex>0))]
 40:    proc  1 (Th:1) 3.6.3.barrier_nonsol.pml:25 (state 2)   [mutex = (mutex-1)]
                mutex = 0
 41:    proc  1 (Th:1) 3.6.3.barrier_nonsol.pml:26 (state 4)   [temp = count]
                Th(1):temp = 4
 42:    proc  1 (Th:1) 3.6.3.barrier_nonsol.pml:27 (state 5)   [count = (temp+1)]
                count = 5

...
```

3.6.2 Barrier non-solution (3.6.3.barrier_nonsol.pml)

```
...

 43:    proc  5 (Th:1) 3.6.3.barrier_nonsol.pml:31 (state 8)   [((count==5))]
 44:    proc  5 (Th:1) 3.6.3.barrier_nonsol.pml:32 (state 9)   [bar[i] = 1]
                bar[0] = 0
                bar[1] = 0
                bar[2] = 0
                bar[3] = 0
                bar[4] = 0
                bar[5] = 1
 45:    proc  5 (Th:1) 3.6.3.barrier_nonsol.pml:33 (state 10)  [assert(((((!(bar[1])||!
(bar[2]))||!(bar[3]))||!(bar[4]))||!(bar[5])))]
 46:    proc  5 (Th:1) 3.6.3.barrier_nonsol.pml:34 (state 11)  [barrier = (barrier+1)]
                barrier = 1
 47:    proc  5 (Th:1) 3.6.3.barrier_nonsol.pml:37 (state 15)  [((barrier>0))]
 47:    proc  5 (Th:1) 3.6.3.barrier_nonsol.pml:37 (state 16)  [barrier = (barrier-1)]
                barrier = 0
                          Th(5): count = 5
 48:    proc  5 (Th:1) 3.6.3.barrier_nonsol.pml:38 (state 18)  [printf('Th(%d): count = %d\\
n',i,count)]
 49: proc 5 terminates

...
```

3.6.2 Barrier non-solution (3.6.3.barrier_nonsol.pml)

```
...

 50:    proc  4 (Th:1) 3.6.3.barrier_nonsol.pml:31 (state 8)   [((count==5))]
 51:    proc  4 (Th:1) 3.6.3.barrier_nonsol.pml:32 (state 9)   [bar[i] = 1]
                bar[0] = 0
                bar[1] = 0
                bar[2] = 0
                bar[3] = 0
                bar[4] = 1
                bar[5] = 1
 52:    proc  4 (Th:1) 3.6.3.barrier_nonsol.pml:33 (state 10)  [assert(((((!(bar[1])||!
(bar[2]))||!(bar[3]))||!(bar[4]))||!(bar[5]))))]
 53:    proc  4 (Th:1) 3.6.3.barrier_nonsol.pml:34 (state 11)  [barrier = (barrier+1)]
                barrier = 1
 54:    proc  4 (Th:1) 3.6.3.barrier_nonsol.pml:37 (state 15)  [((barrier>0))]
 54:    proc  4 (Th:1) 3.6.3.barrier_nonsol.pml:37 (state 16)  [barrier = (barrier-1)]
                barrier = 0
                      Th(4): count = 5
 55:    proc  4 (Th:1) 3.6.3.barrier_nonsol.pml:38 (state 18)  [printf('Th(%d): count = %d\\
n',i,count)]
 56: proc 4 terminates
...
```

3.6.2 Barrier non-solution (3.6.3.barrier_nonsol.pml)

```
...

 57:    proc  3 (Th:1) 3.6.3.barrier_nonsol.pml:31 (state 8)   [((count==5))]
 58:    proc  3 (Th:1) 3.6.3.barrier_nonsol.pml:32 (state 9)   [bar[i] = 1]
                bar[0] = 0
                bar[1] = 0
                bar[2] = 0
                bar[3] = 1
                bar[4] = 1
                bar[5] = 1
 59:    proc  3 (Th:1) 3.6.3.barrier_nonsol.pml:33 (state 10)  [assert(((((!(bar[1])||!
(bar[2]))||!(bar[3]))||!(bar[4]))||!(bar[5]))))]
 60:    proc  3 (Th:1) 3.6.3.barrier_nonsol.pml:34 (state 11)  [barrier = (barrier+1)]
                barrier = 1
 61:    proc  3 (Th:1) 3.6.3.barrier_nonsol.pml:37 (state 15)  [((barrier>0))]
 61:    proc  3 (Th:1) 3.6.3.barrier_nonsol.pml:37 (state 16)  [barrier = (barrier-1)]
                barrier = 0
                  Th(3): count = 5
 62:    proc  3 (Th:1) 3.6.3.barrier_nonsol.pml:38 (state 18)  [printf('Th(%d): count = %d\\
n',i,count)]
 63: proc 3 terminates

...
```

3.6.2 Barrier non-solution (3.6.3.barrier_nonsol.pml)

```
...

 64:    proc  2 (Th:1) 3.6.3.barrier_nonsol.pml:31 (state 8)   [((count==5))]
 65:    proc  2 (Th:1) 3.6.3.barrier_nonsol.pml:32 (state 9)   [bar[i] = 1]
                bar[0] = 0
                bar[1] = 0
                bar[2] = 1
                bar[3] = 1
                bar[4] = 1
                bar[5] = 1
 66:    proc  2 (Th:1) 3.6.3.barrier_nonsol.pml:33 (state 10)  [assert(((((!(bar[1])||!
(bar[2]))||!(bar[3]))||!(bar[4]))||!(bar[5])))]
 67:    proc  2 (Th:1) 3.6.3.barrier_nonsol.pml:34 (state 11)  [barrier = (barrier+1)]
                barrier = 1
 68:    proc  2 (Th:1) 3.6.3.barrier_nonsol.pml:37 (state 15)  [((barrier>0))]
 68:    proc  2 (Th:1) 3.6.3.barrier_nonsol.pml:37 (state 16)  [barrier = (barrier-1)]
                barrier = 0
              Th(2): count = 5
 69:    proc  2 (Th:1) 3.6.3.barrier_nonsol.pml:38 (state 18)  [printf('Th(%d): count = %d\\
n',i,count)]
 70: proc 2 terminates

...
```

3.6.2 Barrier non-solution (3.6.3.barrier_nonsol.pml)

```
...

 71:    proc  1 (Th:1) 3.6.3.barrier_nonsol.pml:28 (state 6)   [mutex = (mutex+1)]
                mutex = 1
 72:    proc  1 (Th:1) 3.6.3.barrier_nonsol.pml:29 (state 7)   [bar[i] = 0]
 73:    proc  1 (Th:1) 3.6.3.barrier_nonsol.pml:31 (state 8)   [((count==5))]
 74:    proc  1 (Th:1) 3.6.3.barrier_nonsol.pml:32 (state 9)   [bar[i] = 1]
                bar[0] = 0
                bar[1] = 1
                bar[2] = 1
                bar[3] = 1
                bar[4] = 1
                bar[5] = 1
spin: 3.6.3.barrier_nonsol.pml:33, Error: assertion violated
spin: text of failed assertion: assert(((((!(bar[1])||!(bar[2]))||!(bar[3]))||!(bar[4]))||!
(bar[5])))
 75:    proc  1 (Th:1) 3.6.3.barrier_nonsol.pml:33 (state 10)  [assert(((((!(bar[1])||!
(bar[2]))||!(bar[3]))||!(bar[4]))||!(bar[5])))]
spin: trail ends after 75 steps

...
```

3.6.2 Barrier non-solution (3.6.3.barrier_nonsol.pml)

```
...


#processes: 2
                count = 5
                mutex = 1
                barrier = 0
                bar[0] = 0
                bar[1] = 1
                bar[2] = 1
                bar[3] = 1
                bar[4] = 1
                bar[5] = 1
 75:    proc  1 (Th:1) 3.6.3.barrier_nonsol.pml:34 (state 11)
 75:    proc  0 (:init::1) 3.6.3.barrier_nonsol.pml:51 (state 11) <valid end state>
6 processes created
```

3.6.4 Barrier solution (3.6.4a.barrier_sol.pml)

```
$ cat -n 3.6.4a.barrier_sol.pml | expand

     1  /*  The Little Book of Semaphores (2.2.1)
     2      by A. Downey
     3
     4      Chapter 3. Basic synchronization patterns
     5
     6      3.6 Barrier
     7      3.6.4 Barrier solution
     8
     9      vk, 2017
    10  */
    11
    12  #define THREADS 5    /* value for threads number */
    13  #define N       5    /* value for barrier limit */
    14
    15  #define wait(sem)   atomic { sem > 0; sem-- }
    16  #define signal(sem) sem++
    17
...
```

3.6.4 Barrier solution (3.6.4a.barrier_sol.pml)

```
...
byte count=0, mutex=1, barrier=0    /* barrier is locked */

proctype Th(byte i) {
    byte temp

    do
    ::  wait(mutex)
            temp=count
            count=temp+1
        signal(mutex)
        if
        :: count == N ->
            signal(barrier)
        :: else
        fi
        wait(barrier)
        printf("Th(%d): count = %d\n",i,count)
        signal(barrier)
        break    /* one only iteration */
    od
}

...
```

3.6.4 Barrier solution (3.6.4a.barrier_sol.pml)

```
...
init {
    byte i

    atomic {
        for (i: 1 .. THREADS) {
            run Th(i)
        }
    }
    _nr_pr == 1 ->
        assert(barrier != 0)    /* barrier (turnstile) is open! */
        printf("barrier = %d\n",barrier)
}
...
```

3.6.4 Barrier solution (3.6.4a.barrier_sol.pml)

```
$ spin 3.6.4a.barrier_sol.pml | expand
                  Th(3): count = 5
                      Th(4): count = 5
              Th(2): count = 5
                          Th(5): count = 5
          Th(1): count = 5
      barrier = 1
6 processes created


$ spin 3.6.4a.barrier_sol.pml | expand
                      Th(4): count = 5
                  Th(3): count = 5
          Th(1): count = 5
              Th(2): count = 5
                          Th(5): count = 5
      barrier = 1
6 processes created
```

3.6.4 Barrier solution (3.6.4a.barrier_sol.pml)

```
$ spin 3.6.4a.barrier_sol.pml | expand
                      Th(4): count = 5
                  Th(3): count = 5
              Th(2): count = 5
                          Th(5): count = 5
          Th(1): count = 5
      barrier = 2
6 processes created
```

3.6.4 Barrier solution (3.6.4b.barrier_sol.pml)

```
...
init {
    byte i

    atomic {
        for (i: 1 .. THREADS) {
            run Th(i)
        }
    }
    _nr_pr == 1 ->
        assert(0 < barrier && barrier < 5)
        printf("barrier = %d\n",barrier)
}
...
```

3.6.4 Barrier solution (3.6.4b.barrier_sol.pml)

```
$ spin -run 3.6.4b.barrier_sol.pml | expand
pan:1: assertion violated ((0<barrier)&&(barrier<5)) (at depth 70)
pan: wrote 3.6.4b.barrier_sol.pml.trail

(Spin Version 6.4.8 -- 2 March 2018)
Warning: Search not completed
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 64 byte, depth reached 72, errors: 1
...
```

3.6.4 Barrier solution (3.6.4b.barrier_sol.pml)

```
$ spin -t -p -g -l 3.6.4b.barrier_sol.pml | expand
using statement merging
  1:    proc  0 (:init::1) 3.6.4b.barrier_sol.pml:44 (state 1)  [i = 1]
                :init:(0):i = 1
  2:    proc  0 (:init::1) 3.6.4b.barrier_sol.pml:44 (state 2)  [((i<=5))]
Starting Th with pid 1
  3:    proc  0 (:init::1) 3.6.4b.barrier_sol.pml:45 (state 3)  [(run Th(i))]
  4:    proc  0 (:init::1) 3.6.4b.barrier_sol.pml:44 (state 4)  [i = (i+1)]
                :init:(0):i = 2
  5:    proc  0 (:init::1) 3.6.4b.barrier_sol.pml:44 (state 2)  [((i<=5))]
Starting Th with pid 2
...
                :init:(0):i = 6
 17:    proc  0 (:init::1) 3.6.4b.barrier_sol.pml:46 (state 5)  [else]
 18:    proc  0 (:init::1) 3.6.4b.barrier_sol.pml:46 (state 6)  [goto :b1]
 19:    proc  0 (:init::1) 3.6.4b.barrier_sol.pml:46 (state 9)  [break]

 20:    proc  5 (Th:1) 3.6.4b.barrier_sol.pml:24 (state 1)      [((mutex>0))]
 20:    proc  5 (Th:1) 3.6.4b.barrier_sol.pml:24 (state 2)      [mutex = (mutex-1)]
                mutex = 0
 21:    proc  5 (Th:1) 3.6.4b.barrier_sol.pml:25 (state 4)      [temp = count]
                Th(5):temp = 0
 22:    proc  5 (Th:1) 3.6.4b.barrier_sol.pml:26 (state 5)      [count = (temp+1)]
                count = 1
 23:    proc  5 (Th:1) 3.6.4b.barrier_sol.pml:27 (state 6)      [mutex = (mutex+1)]
                mutex = 1

...
```

3.6.4 Barrier solution (3.6.4b.barrier_sol.pml)

```
...
 24:    proc  4 (Th:1) 3.6.4b.barrier_sol.pml:24 (state 1)      [((mutex>0))]
 24:    proc  4 (Th:1) 3.6.4b.barrier_sol.pml:24 (state 2)      [mutex = (mutex-1)]
                mutex = 0
 25:    proc  4 (Th:1) 3.6.4b.barrier_sol.pml:25 (state 4)      [temp = count]
                Th(4):temp = 1
 26:    proc  4 (Th:1) 3.6.4b.barrier_sol.pml:26 (state 5)      [count = (temp+1)]
                count = 2
 27:    proc  4 (Th:1) 3.6.4b.barrier_sol.pml:27 (state 6)      [mutex = (mutex+1)]
                mutex = 1

 28:    proc  3 (Th:1) 3.6.4b.barrier_sol.pml:24 (state 1)      [((mutex>0))]
 28:    proc  3 (Th:1) 3.6.4b.barrier_sol.pml:24 (state 2)      [mutex = (mutex-1)]
                mutex = 0
 29:    proc  3 (Th:1) 3.6.4b.barrier_sol.pml:25 (state 4)      [temp = count]
                Th(3):temp = 2
 30:    proc  3 (Th:1) 3.6.4b.barrier_sol.pml:26 (state 5)      [count = (temp+1)]
                count = 3
 31:    proc  3 (Th:1) 3.6.4b.barrier_sol.pml:27 (state 6)      [mutex = (mutex+1)]
                mutex = 1

 32:    proc  2 (Th:1) 3.6.4b.barrier_sol.pml:24 (state 1)      [((mutex>0))]
 32:    proc  2 (Th:1) 3.6.4b.barrier_sol.pml:24 (state 2)      [mutex = (mutex-1)]
                mutex = 0
 33:    proc  2 (Th:1) 3.6.4b.barrier_sol.pml:25 (state 4)      [temp = count]
                Th(2):temp = 3
 34:    proc  2 (Th:1) 3.6.4b.barrier_sol.pml:26 (state 5)      [count = (temp+1)]
                count = 4
 35:    proc  2 (Th:1) 3.6.4b.barrier_sol.pml:27 (state 6)      [mutex = (mutex+1)]
                mutex = 1
...
```

3.6.4 Barrier solution (3.6.4b.barrier_sol.pml)

```
...
 36:     proc  1 (Th:1) 3.6.4b.barrier_sol.pml:24 (state 1)      [((mutex>0))]
 36:     proc  1 (Th:1) 3.6.4b.barrier_sol.pml:24 (state 2)      [mutex = (mutex-1)]
                mutex = 0
 37:     proc  1 (Th:1) 3.6.4b.barrier_sol.pml:25 (state 4)      [temp = count]
                Th(1):temp = 4
 38:     proc  1 (Th:1) 3.6.4b.barrier_sol.pml:26 (state 5)      [count = (temp+1)]
                count = 5

 39:     proc  5 (Th:1) 3.6.4b.barrier_sol.pml:29 (state 7)      [((count==5))]
 40:     proc  5 (Th:1) 3.6.4b.barrier_sol.pml:30 (state 8)      [barrier = (barrier+1)]
                barrier = 1
 41:     proc  5 (Th:1) 3.6.4b.barrier_sol.pml:33 (state 12)     [((barrier>0))]
 41:     proc  5 (Th:1) 3.6.4b.barrier_sol.pml:33 (state 13)     [barrier = (barrier-1)]
                barrier = 0
                          Th(5): count = 5
 42:     proc  5 (Th:1) 3.6.4b.barrier_sol.pml:34 (state 15)     [printf('Th(%d): count = %d\\
n',i,count)]
 43:     proc  5 (Th:1) 3.6.4b.barrier_sol.pml:35 (state 16)     [barrier = (barrier+1)]
                barrier = 1
 44: proc 5 terminates

...
```

3.6.4 Barrier solution (3.6.4b.barrier_sol.pml)

```
...
 45:     proc  4 (Th:1) 3.6.4b.barrier_sol.pml:29 (state 7)      [((count==5))]
 46:     proc  4 (Th:1) 3.6.4b.barrier_sol.pml:30 (state 8)      [barrier = (barrier+1)]
                barrier = 2
 47:     proc  4 (Th:1) 3.6.4b.barrier_sol.pml:33 (state 12)     [((barrier>0))]
 47:     proc  4 (Th:1) 3.6.4b.barrier_sol.pml:33 (state 13)     [barrier = (barrier-1)]
                barrier = 1
                      Th(4): count = 5
 48:     proc  4 (Th:1) 3.6.4b.barrier_sol.pml:34 (state 15)     [printf('Th(%d): count = %d\\
n',i,count)]
 49:     proc  4 (Th:1) 3.6.4b.barrier_sol.pml:35 (state 16)     [barrier = (barrier+1)]
                barrier = 2
 50: proc 4 terminates


 51:     proc  3 (Th:1) 3.6.4b.barrier_sol.pml:29 (state 7)      [((count==5))]
 52:     proc  3 (Th:1) 3.6.4b.barrier_sol.pml:30 (state 8)      [barrier = (barrier+1)]
                barrier = 3
 53:     proc  3 (Th:1) 3.6.4b.barrier_sol.pml:33 (state 12)     [((barrier>0))]
 53:     proc  3 (Th:1) 3.6.4b.barrier_sol.pml:33 (state 13)     [barrier = (barrier-1)]
                barrier = 2
                  Th(3): count = 5
 54:     proc  3 (Th:1) 3.6.4b.barrier_sol.pml:34 (state 15)     [printf('Th(%d): count = %d\\
n',i,count)]
 55:     proc  3 (Th:1) 3.6.4b.barrier_sol.pml:35 (state 16)     [barrier = (barrier+1)]
                barrier = 3
 56: proc 3 terminates

...
```

3.6.4 Barrier solution (3.6.4b.barrier_sol.pml)

```
...
 57:    proc  2 (Th:1) 3.6.4b.barrier_sol.pml:29 (state 7)      [((count==5))]
 58:    proc  2 (Th:1) 3.6.4b.barrier_sol.pml:30 (state 8)      [barrier = (barrier+1)]
                barrier = 4
 59:    proc  2 (Th:1) 3.6.4b.barrier_sol.pml:33 (state 12)     [((barrier>0))]
 59:    proc  2 (Th:1) 3.6.4b.barrier_sol.pml:33 (state 13)     [barrier = (barrier-1)]
                barrier = 3
              Th(2): count = 5
 60:    proc  2 (Th:1) 3.6.4b.barrier_sol.pml:34 (state 15)     [printf('Th(%d): count = %d\\
n',i,count)]
 61:    proc  2 (Th:1) 3.6.4b.barrier_sol.pml:35 (state 16)     [barrier = (barrier+1)]
                barrier = 4
 62: proc 2 terminates


 63:    proc  1 (Th:1) 3.6.4b.barrier_sol.pml:27 (state 6)      [mutex = (mutex+1)]
                mutex = 1
 64:    proc  1 (Th:1) 3.6.4b.barrier_sol.pml:29 (state 7)      [((count==5))]
 65:    proc  1 (Th:1) 3.6.4b.barrier_sol.pml:30 (state 8)      [barrier = (barrier+1)]
                barrier = 5
 66:    proc  1 (Th:1) 3.6.4b.barrier_sol.pml:33 (state 12)     [((barrier>0))]
 66:    proc  1 (Th:1) 3.6.4b.barrier_sol.pml:33 (state 13)     [barrier = (barrier-1)]
                barrier = 4
          Th(1): count = 5
 67:    proc  1 (Th:1) 3.6.4b.barrier_sol.pml:34 (state 15)     [printf('Th(%d): count = %d\\
n',i,count)]
 68:    proc  1 (Th:1) 3.6.4b.barrier_sol.pml:35 (state 16)     [barrier = (barrier+1)]
                barrier = 5
 69: proc 1 terminates
...
```

3.6.4 Barrier solution (3.6.4b.barrier_sol.pml)

```
...
 70:    proc  0 (:init::1) 3.6.4b.barrier_sol.pml:48 (state 11) [((_nr_pr==1))]
spin: 3.6.4b.barrier_sol.pml:49, Error: assertion violated
spin: text of failed assertion: assert(((0<barrier)&&(barrier<5)))
 71:    proc  0 (:init::1) 3.6.4b.barrier_sol.pml:49 (state 12)
[assert(((0<barrier)&&(barrier<5)))]
spin: trail ends after 71 steps
#processes: 1
                count = 5
                mutex = 1
                barrier = 5
 71:    proc  0 (:init::1) 3.6.4b.barrier_sol.pml:50 (state 13)
6 processes created
```

3.6.5 Bad barrier solution (3.6.5.bad_barrier.pml)

```
$ cat -n 3.6.5.bad_barrier.pml | expand

     1  /*  The Little Book of Semaphores (2.2.1)
     2      by A. Downey
     3
     4      Chapter 3. Basic synchronization patterns
     5
     6      3.6 Barrier
     7      3.6.5 Bad barrier solution (deadlock)
     8
     9      vk, 2017
    10  */
    11
    12  #define THREADS 3    /* value for threads number */
    13  #define N       3    /* value for barrier limit */
    14
    15  #define wait(sem)   atomic { sem > 0; sem-- }
    16  #define signal(sem) sem++
    17
    18  byte count=0, mutex=1, barrier=0    /* barrier is locked */
    19
...
```

3.6.5 Bad barrier solution (3.6.5.bad_barrier.pml)

```
...
    20  proctype Th(byte i) {
    21      byte temp
    22
    23  rendezvous:
    24      do
    25      ::  wait(mutex)
    26              temp=count
    27              count=temp+1
    28              if
    29              :: count == N ->
    30                  signal(barrier)
    31              :: else
    32              fi
    33              wait(barrier)
    34              printf("Th(%d): count = %d\n",i,count)
    35              signal(barrier)
    36          signal(mutex)
    37          break    /* one only iteration */
    38      od
    39  critical_point:
    40  }
...
```

3.6.5 Bad barrier solution (3.6.5.bad_barrier.pml)

```
...
    41
    42  init {
    43      byte i
    44
    45      atomic {
    46          for (i: 1 .. THREADS) {
    47              run Th(i)
    48          }
    49      }
    50  }

$ spin 3.6.5.bad_barrier.pml | expand
      timeout
#processes: 4
                count = 1
                mutex = 0
                barrier = 0
 19:    proc  3 (Th:1) 3.6.5.bad_barrier.pml:24 (state 18)
 19:    proc  2 (Th:1) 3.6.5.bad_barrier.pml:24 (state 18)
 19:    proc  1 (Th:1) 3.6.5.bad_barrier.pml:33 (state 13)
 19:    proc  0 (:init::1) 3.6.5.bad_barrier.pml:50 (state 11) <valid end
state>
4 processes created
```