

## Question 1:

1. **How many ways can the three statements of two identical processes be interleaved?**
  - 20 different possible interleave variations
2. **How many ways can the four statements of three identical processes be interleaved?**
  - 34'650 different possible interleave variations
3. **How many ways can the n statements of m identical processes be interleaved?**
  - For m different identical processes, each with n different statements, the number of possible different interleave variations can be calculated using:
    - $\#Variations = (mxn)!/(n!)^m$

## Question 2:

Using Promela/Spin show that the Bakery lock has the following desirable properties: safety, deadlock free, liveness and starvation free.

The four properties that are required to be proved in this assignment are:

1. Safety/Mutual Exclusion
2. Deadlock free
3. Liveness
4. Starvation Free

### Implementation of the Bakery Lock:

The below screenshot of code of the bakery lock is implemented using the Promela programming language. This code does not contain any attempts to try and prove the above properties.

```

1 int MAX_NUM_THREADS = 2;
2 byte choosingThread[MAX_NUM_THREADS];
3 byte ticketNumber[MAX_NUM_THREADS];
4
5 proctype customer(){
6     byte id = _pid - 1;
7
8     int numIters;
9     for(numIters : 0 .. 2){
10         choosingThread[id]=1; // Threads request lock
11
12         int i, max=0;
13
14         for(i : 0 .. (MAX_NUM_THREADS - 1)){ // loop across all threads that are active.
15             // max is set to 0, if there is a ticketNumber that is greater than max, max =
16             ticketNumber
17                 if
18                     :: ticketNumber[i] > max -> max = ticketNumber[i];
19                     :: else;
20                     fi;
21         }
22
23         // Take a ticket
24         ticketNumber[id] = max + 1;
25         choosingThread[id] = 0;
26
27         int j;
28         for(j : 0 .. (MAX_NUM_THREADS - 1)){
29             // Wait for our turn to come!
30             // if choosing[j] == 0, then we are on another thread and we must
31             // wait until it is our turn (choosingThread[j] == 1) threads turn to execute
32             do
33                 :: (choosingThread[j] == 0) -> break;
34             od;
35
36             ((ticketNumber[j] == 0) || ((ticketNumber[j] >= ticketNumber[id]) &&
37             ((ticketNumber[j] != ticketNumber[id] || (j >= id)))));
38         }
39
40         // This is the critical section of the code!
41         // It will do some - call a function/ update a data structure ETC
42
43         // Release the lock by clearing our ticket in ticketNumber
44         ticketNumber[id] = 0;
45     }
46     printf("Process %d has competed %d iterations of this loop", id, numIters);
47 }
48
49 init {
50     run customer();
51     run customer();
52     _nr_pr == 1;
53 }

```

**1. Safety / Mutual Exclusion:**

Safety was determined by adding in an extra variable; *test\_in\_critical\_section*, which was incremented in the critical section of thread execution in order to assert safety - that mutual exclusion is not violated.

```

37         // This is the critical section of the code!
38         // It will do some - call a function/ update a data stucture ETC
39         test_in_critical_section++;
40         assert(test_in_critical_section == 1);
41         test_in_critical_section--;
42         // Release the lock by clearing our ticket in ticketNumber

```

Since the assertion is not violated - it was concluded that that there was only 1 thread accessing the critical section at any time - and hence that the Safety is ensured and mutual exclusion is no violated in the bakery lock.

Violation of safety in the bakery lock is achievable by removing line 10:

```

8         int numIters;
9         for(numIters : 0 .. 2){
10             // choosingThread[id]=1; // Threads request lock
11

```

Safety was violated here since neither of the threads were able to identify that the other thread had already requested the lock - which was the purpose of *choosingThread[id]* - this resulted in both threads accessing the *test\_in\_critical\_section* variable in the critical section and caused the assert to fail.

## Advanced Computer Architecture

## Tutorial 1 - Question 2

```

1 int MAX_NUM_THREADS = 2;
2 byte choosingThread[NUM_THREADS];
3 byte ticketNumber[NUM_THREADS];
4 int test_in_critical_section;
5
6 proctype customer(){
7     byte id = _pid - 1;
8     int numIters;
9     for(numIters : 0 .. 2){
10         // Comment the line below in order to break safety/mutual exclusion property
11         choosing[id] = 1; // Threads request lock
12
13         int i, max = 0;
14
15         for(i : 0 .. (NUM_THREADS - 1)){ // loop across all threads that are active.
16             if // max is set to 0, if there is a ticketNumber that is greater than max, max =
ticketNumber
                :: ticketNumber[i] > max -> max = ticketNumber[i];
17             :: else;
18             fi;
19         }
20
21         // Take a ticket
22         ticketNumber[id] = max + 1;
23         choosing[id] = 0;
24
25         int j;
26         for(j : 0 .. (NUM_THREADS - 1)){
27             do
28                 :: (choosing[j] == 0) -> break;
29             od;
30
31             ((ticketNumber[j] == 0) || ((ticketNumber[j] >= ticketNumber[id]) && ((ticketNumber[j] != ticketNumber[id]
32             || (j >= id)))));
33         }
34
35         // This is the critical section of the code!
36         // It will do some - call a function/ update a data structure ETC
37         test_in_critical_section++;
38         assert(test_in_critical_section);
39         test_in_critical_section--;
40         // Release the lock by clearing our ticket in ticketNumber
41         ticketNumber[id] = 0;
42     }
43     printf("Process %d has competed %d iterations of this loop", id, numIters);
44 }
45
46 init{
47     run customer();
48     run customer();
49     (_nr_pr == 1);
50 }

```

Breandan Kerin 1431016 27th September 2018

# Advanced Computer Architecture

## Tutorial 1 - Question 2

Activities ISPIN

Thu 12:35  
bakery\_Lock.pml

Spin Version 6.4.6 -- 2 December 2016 -- ISPIN Version 1.1.4 -- 27 November 2014

FileView Simulate / Replay Verification **Swarm Run** <Help> Save Session Restore Session <Quit>

Safety	Storage Mode	Search Mode
<input checked="" type="checkbox"/> safety	<input checked="" type="checkbox"/> exhaustive	<input checked="" type="checkbox"/> depth-first search
<input checked="" type="checkbox"/> + invalid endstates (deadlock)	<input type="checkbox"/> + minimized automata (slow)	<input checked="" type="checkbox"/> + partial order reduction
<input checked="" type="checkbox"/> + assertion violations	<input type="checkbox"/> + collapse compression	<input type="checkbox"/> + bounded context switching
<input type="checkbox"/> + x/xs assertions	<input type="checkbox"/> hash-compact <input type="checkbox"/> bistate/supertace	with bound: 10
<input type="checkbox"/> non-progress cycles	<input type="checkbox"/> do not use a never claim or tl property	<input type="checkbox"/> + iterative search for short trail
<input type="checkbox"/> acceptance cycles	<input type="checkbox"/> use claim	<input checked="" type="checkbox"/> breadth-first search
<input type="checkbox"/> enforce weak fairness constraint	claim name (opt):	<input checked="" type="checkbox"/> + partial order reduction
	Run	<input checked="" type="checkbox"/> report unreachable code
	Stop	Save Result In:
		pan.out

```
1 int MAX_NUM_THREADS = 2;
2 byte choosingThread[MAX_NUM_THREADS];
3 byte ticketNumber[MAX_NUM_THREADS];
4
5 proctype customer()
6   byte id = _pid - 1;
7
8   int numbers;
9   for(numbers : 0..2){
10     choosingThread[id]=1; // Threads request lock
11
12     int i, max=0;
13
14     for(i : 0..(MAX_NUM_THREADS - 1)){ // loop across all threads that are active.
15       // max is set to 0, if there is a ticketNumber that is greater than max, max = ticketNumber
16       if (ticketNumber[i] > max -> max = ticketNumber[i];
17       :: ticketNumber[i] = max + 1;
18       :: else;
19       fi;
20     }
21
22     // Take a ticket
23     ticketNumber[id] = max + 1;
24     choosingThread[id] = 0;
25
26     int i;
27     for(i : 0..(MAX_NUM_THREADS - 1)){
28       // Wait for our turn to come!
29       // If i crossing[i] == 0, then we are on another thread and we must
30       // wait until it is our turn (choosingThread[i] == 1) threads turn to execute
31       do
32         :: (choosingThread[i] == 0) -> break;
33       od;
34
35       ((ticketNumber[i] == 0) || (ticketNumber[i] >= ticketNumber[id]) && (ticketNumber[i]
36         = ticketNumber[id] || (i >= id)))));
37     }
38     // This is the critical section of the code!
```

no errors found -- and you verify all claims?

spin -a bakery\_Lock.pml  
spin: bakery\_Lock.pml:37: Error: syntax error saw '}' = 125  
gcc -DMEMORY=1024 -O2 -DXUSAFE -DSAFETY -DNOCLAIM -w -o pan pan.c  
/pan -m10000  
Pid: 28988

(Spin Version 6.4.6 -- 2 December 2016)  
+ Partial Order Reduction

Full statespace search for:  
never claim  
assertion violations +  
cycle checks - (disabled by -DSAFETY)  
invalid end states +

State-vector 68 byte, depth reached 146, errors: 0  
17351 states, stored  
7006 states, matched  
24357 transitions (= stored+matched)  
0 atomic steps  
hash conflicts: 0 (resolved)

Stats on memory usage (in Megabytes):  
1.589 equivalent memory usage for states (stored+(State-vector + overhead))  
1.588 actual memory usage for states (compression: 98.06%)  
state-vector as stored = 66 byte + 28 byte overhead  
128.000 memory used for hash table (-w24)  
0.534 memory used for DFS stack (-m10000)  
129.999 total actual memory usage

unreached in procype customer  
(0 of 44 states)  
unreached in init  
(0 of 4 states)  
pan: elapsed time 0.01 seconds  
No errors found -- did you verify all claims?

Show Error Trapping Options Show Advanced Parameter Settings



Breandan Kerin 1431016 27th September 2018  
Advanced Computer Architecture  
Tutorial 1 - Question 2

Activities | Ispim | Thu 12:37 | bakery\_Lock.pml | Spin Version 6.4.6 -- 2 December 2016 :: Ispim Version 1.1.4 -- 27 November 2014

File View | Simulate / Replay | Verification | System Run | <Help> | Save Session | Feature Session | <Quit>

Mode: Random, with seed: 123  
Interactive (for resolution of all nondeterminism)  
Guided, with trail: bakery\_Lock.pml:trail | browse  
Initial steps skipped: 0  
Maximum number of steps: 10000  
MSC max text width: 20  
MSC update delay: 25  
Tracked variable: track scaling: Step Backward | Step Forward | Rewind | Stop | (Re)Run

Background command executed:  
spin -p -s -r -X -v -n123 -t -g -u10000 bakery\_Lock.pml

Save in: msc.ps

```
[Variable values, step 242]
customer(1):i = 2
customer(1):j = 2
customer(1):numIters = 3
customer(2):i = 2
customer(2):j = 2
customer(2):numIters = 3
spin: bakery_Lock.pml:36, Error: syntax error saw '}'
ticketNumber[0] = 0
ticketNumber[1] = 0

212: [proc 2 (customer1) bakery_Lock.pml:32 (state 21) [(choosing ThreadID==0)]]
214: proc 1 (customer1) bakery_Lock.pml:27 (state 20) [(k<=(MAX_NUM_THREADS-1))]
216: proc 2 (customer1) bakery_Lock.pml:36 (state 26) [(1)]
217: proc 2 (customer1) bakery_Lock.pml:27 (state 27) [(i=(i+1))]
219: proc 2 (customer1) bakery_Lock.pml:32 (state 20) [(k<=(MAX_NUM_THREADS-1))]
221: proc 1 (customer1) bakery_Lock.pml:32 (state 21) [(choosing ThreadID==0)]
223: proc 2 (customer1) bakery_Lock.pml:32 (state 21) [(choosing ThreadID==0)]
225: proc 1 (customer1) bakery_Lock.pml:36 (state 26) [(1)]
226: proc 1 (customer1) bakery_Lock.pml:27 (state 27) [(i=(i+1))]
227: proc 2 (customer1) bakery_Lock.pml:36 (state 26) [(1)]
229: proc 2 (customer1) bakery_Lock.pml:27 (state 27) [(i=(i+1))]
231: proc 2 (customer1) bakery_Lock.pml:36 (state 28) [false]
232: proc 1 (customer1) bakery_Lock.pml:36 (state 28) [false]
234: proc 1 (customer1) bakery_Lock.pml:42 (state 33) [ticketNumber[i] = 0]
235: proc 1 (customer1) bakery_Lock.pml:39 (state 34) [numIters = (numIters+1)]
236: proc 1 (customer1) bakery_Lock.pml:39 (state 35) [else]
238: proc 0 has competed 3 iterations of this loop240: proc 1 (customer1) bakery_Lock.pml:45 (state 40) [print("Process %d has competed %d iterations of this loop",id,numIters)]
241: proc 2 (customer1) bakery_Lock.pml:42 (state 33) [ticketNumber[i] = 0]
242: proc 2 (customer1) bakery_Lock.pml:39 (state 34) [numIters = (numIters+1)]
244: proc 2 (customer1) bakery_Lock.pml:43 (state 35) [else]
246: proc 2 (customer1) terminates
247: proc 0 (init:1) bakery_Lock.pml:51 (state 3) [(!nr_p==1)]
247: proc 0 (init:1) terminates
3 processes created
```

## 2. Deadlock free

Due to the fact that both threads were able to execute their instructions 3 times, each to completion without reaching an '*invalid end state*' in SPIN, allows me to conclude that the bakery lock implementation is deadlock free, and deadlock was not violated in the bakery lock implementation shown.

It can be noted that deadlock can be induced on the program by removing line 23:

```

22      // Take a ticket
23      ticketNumber[id] = max + 1;
24      // choosingThread[id] = 0;
25
26      int j;
27      for(j : 0 .. (MAX_NUM_THREADS - 1)){
28          // Wait for our turn to come!
29          // if choosing[j] == 0, then we are on another thread and we must
30          // wait until it is our turn (choosingThread[j] == 1) threads turn to execute
31          do
32          :: (choosingThread[j] == 0) -> break;
33          od;
34

```

This means that when the code enters the *do* section of the program - on lines 29-31 as per above, each of the two threads run would eventually enter an infinite loop - and hence break this expression. This is due to the fact that neither thread will be able to progress beyond this point until it knows the *choosingThread[id]* flag for the rest of the live threads has been reset back to 0; *choosingThread[id] = 0*.

Since the threads were not reset to 0, only one thread could enter the critical section and as a result none of the threads will be able to progress beyond this point since they are waiting on each other to no longer require the lock.

It is clear that, since the code had to be altered; removing line 23 - *choosingThread[id] = 0* - in order to cause deadlock to occur, it can be concluded that the bakery lock is deadlock free. From the screen shots below - it can be seen that SPIN detected deadlock due to the - '**invalid end state**' shown below.

## Advanced Computer Architecture

## Tutorial 1 - Question 2

```

1 int MAX_NUM_THREADS = 2;
2 byte choosingThread[NUM_THREADS];
3 byte ticketNumber[NUM_THREADS];
4
5 proctype customer(){
6     byte id = _pid - 1;
7
8     int numIters;
9     for(numIters : 0 .. 2){
10         choosing[id] = 1; // Threads request lock
11
12         int i, max = 0;
13
14         for(i : 0 .. (NUM_THREADS - 1)){ // loop across all threads that are active.
15             if // max is set to 0, if there is a ticketNumber that is greater than max, max =
ticketNumber
16                 :: ticketNumber[i] > max -> max = ticketNumber[i];
17                 :: else;
18                 fi;
19         }
20
21         // Take a ticket
22         ticketNumber[id] = max + 1;
23         // choosing[id] = 0; // To remove deadlock from program - remove comment from this line!
24
25         int j;
26         for(j : 0 .. (NUM_THREADS - 1)){
27             // Wait for our turn to come!
28             // if choosing[j] == 0, then we are on another thread and we must wait until it is our turn (choosing[j]
== 1) threads turn to execute
29             do
30                 :: (choosing[j] == 0) -> break;
31             od;
32
33             ((ticketNumber[j] == 0) || ((ticketNumber[j] >= ticketNumber[id]) && ((ticketNumber[j] != ticketNumber[id]
|| (j >= id)))));
34         }
35
36         // This is the critical section of the code!
37         // It will do some - call a function/ update a data structure ETC
38
39         // Release the lock by clearing our ticket in ticketNumber
40         ticketNumber[id] = 0;
41     }
42     printf("Process %d has competed %d iterations of this loop", id, numIters);
43 }
44
45 init{
46     run customer();
47     run customer();
48     (_nr_pr == 1);
49 }
50

```



Breandan Kerin 1431016 27th September 2018  
Advanced Computer Architecture  
Tutorial 1 - Question 2

Activities lsph Thu 12:41  bakery\_lock.pml

Spin Version 6.4.6 -- 2 December 2016 :: lsph Version 1.1.4 -- 27 November 2014

Edit/View Simulate / Replay Verification <Help> Save Session Restore Session <Quit>

Safety	Storage Mode	Search Mode
<input checked="" type="checkbox"/> safety <input checked="" type="checkbox"/> + invalid endstates (deadlock) <input checked="" type="checkbox"/> + assertion violations <input type="checkbox"/> + x/rxs assertions	<input type="checkbox"/> exhaustive <input type="checkbox"/> + minimized automata (slow) <input type="checkbox"/> + collapse compression <input type="checkbox"/> hash-compact <input type="checkbox"/> bistate/supertace	<input type="checkbox"/> depth-first search <input checked="" type="checkbox"/> + partial order reduction <input type="checkbox"/> + bounded context switching with bound: 0
<input type="checkbox"/> non-progress cycles <input type="checkbox"/> acceptance cycles <input type="checkbox"/> enforce weak fairness constraint	<input type="checkbox"/> do not use a never claim or its property <input type="checkbox"/> use claim claim name (opt):	<input type="checkbox"/> + iterative search for short trail <input type="checkbox"/> breadth-first search <input checked="" type="checkbox"/> + partial order reduction <input checked="" type="checkbox"/> report unreachable code

```
1 int MAX_NUM_THREADS = 2;
2 byte choosingThread[MAX_NUM_THREADS];
3 byte ticketNumber[MAX_NUM_THREADS];
4
5 prototype customer()
6     byte id = _pid - 1;
7
8     int numbers;
9     for(numbers : 0..2){
10         choosingThread[id]=1; // Thread's request lock
11
12         int l, max=0;
13
14         for(i : 0..(MAX_NUM_THREADS - 1)){ // loop across all threads that are active.
15             // max is set to 0, if there is a ticketNumber that is greater than max, max = ticketNumber
16             if
17                 :: ticketNumber[i] > max -> max = ticketNumber[i];
18             :: else;
19                 fi;
20         }
21
22         // Take a ticket
23         ticketNumber[id] = max + 1;
24         // choosingThread[id] = 0;
25
26         int i;
27         for(i : 0..(MAX_NUM_THREADS - 1)){
28             // Wait for our turn to come!
29             // If choosing[i] == 0, then we are on another thread and we must
30             // wait until it is our turn (choosingThread[i] == 1) threads turn to execute
31             do
32                 :: (choosingThread[i] == 0) -> break;
33             od;
34
35             ((ticketNumber[i] == 0) || ((ticketNumber[i] >= ticketNumber[id] && (ticketNumber[i] != ticketNum
36             ber[id] || (i >= id)))));
37         }
38     }
39     // This is the critical section of the code!
```

pan: elapsed time 0.01 seconds  
No errors found -- did you verify all claims?  
spin -a bakery\_lock.pml  
spin: bakery\_lock.pml:37: Error: syntax error saw '}' = 125  
gcc -DMEMLINE=1024 -O2 -DXUSAFE -DSAFETY -DNOCLAIM -w -o pan pan.c  
./pan -m10000  
Pid: 29124  
pan.t: invalid end state (at depth 33)  
pan: wrote bakery\_lock.pml.trail  
(Spin Version 6.4.6 -- 2 December 2016)  
Warning: Search not completed  
+ Partial Order Reduction  
Full statespace search for:  
- never claim  
- assertion violations +  
- cycle checks - (disabled by -DSAFETY)  
- invalid end states +  
State-vector 68 bytes, depth reached 34, errors: 1  
35 states, stored  
0 states, matched  
35 transitions (= stored+matched)  
0 atomic steps  
hash conflicts: 0 (resolved)  
Stats on memory usage (in Megabytes):  
0.003 equivalent memory usage for states (stored\*(State-vector + overhead))  
0.288 actual memory usage for states  
128.000 memory used for hash table (-w24)  
0.534 memory used for DFS stack (-m10000)  
128.730 total actual memory usage  
pan: To replay the error-trail, goto Simulate/Replay and select "Run"

Breandan Kerin 1431016 27th September 2018  
Advanced Computer Architecture  
Tutorial 1 - Question 2

Activities | 1spin

Thu 12:43

bakery.Lock.pml

Spin Version 6.4.6 -- 2 December 2016 :: 1spin Version 1.1.4 -- 27 November 2014

Backround command executed:  
spin -p -s -r -X -v -n123 -l -g -k bakery.Lock.pml:trail -u10000 bakery.Lock.pml

Save in: msc.ps

Mode: Random, with seed: 123  
Interactive (for resolution of all nondeterminism)  
Guided, with trail: bakery.Lock.pml:trail  
Initial steps skipped: 0  
Maximum number of steps: 10000  
MSC max text width: 20  
MSC update delay: 25  
Tracked variable:  
Track Data Values (this can be slow): track scaling: Step Backward

Output Filtering (reg. exps.):  
process ids:  
queue ids:  
var names:  
Step Forward  
Step Backward

(re)Run  
Stop  
Rewind

```
26 int i;  
27 for(i : 0..(MAX_NUM_THREADS - 1))  
28 // Wait for our turn to come!  
29 // If crossing[i] == 0, then we are on another thread and we must  
30 // wait until it is our turn (choosingThread[i] == 1) threads turn to execute  
31 do  
32 :: (choosingThread[i] == 0) -> break;  
33 od;  
34  
35 ((ticketNumber[i] == 0) && (ticketNumber[i] != ticketNumber[i]))  
36 }  
37  
38 // This is the critical section of the code!  
39 // It will do some - call a function/ update a data structure ETC  
40  
41 // Release the lock by clearing our ticket in ticketNumber  
42 ticketNumber[i] = 0;  
43 }  
44  
45  
46 printf("Process %d had completed %d iterations of this loop" %d, numIter);
```

[Variable values, step 34]

```
MAX_NUM_THREADS = 2  
choosingThread[0] = 1  
choosingThread[1] = 1  
customer(1):i = 2  
customer(1):j = 0  
customer(1):max = 1  
customer(1):numIters = 0  
customer(2):i = 2  
customer(2):j = 1  
customer(2):max = 0  
customer(2):numIters = 0  
spin: bakery.Lock.pml:37, Error: syntax error saw ''  
ticketNumber[0] = 2  
ticketNumber[1] = 1
```

23: proc 1 (customer:1) bakery.Lock.pml:14 (state 6) [(i = 0)]  
24: proc 1 (customer:1) bakery.Lock.pml:18 (state 10) [(i = 0)]  
25: proc 1 (customer:1) bakery.Lock.pml:18 (state 10) [(i = 0)]  
26: proc 1 (customer:1) bakery.Lock.pml:14 (state 13) [(i = 0+1)]  
27: proc 1 (customer:1) bakery.Lock.pml:17 (state 8) [(ticketNumber[i] > max)]  
28: proc 1 (customer:1) bakery.Lock.pml:17 (state 8) [(ticketNumber[i] > max)]  
29: proc 1 (customer:1) bakery.Lock.pml:17 (state 8) [(ticketNumber[i] > max)]  
30: proc 1 (customer:1) bakery.Lock.pml:14 (state 13) [(i = 0+1)]  
31: proc 1 (customer:1) bakery.Lock.pml:20 (state 14) [else]  
32: proc 1 (customer:1) bakery.Lock.pml:23 (state 19) [ticketNumber[i] = (max+1)]  
33: proc 1 (customer:1) bakery.Lock.pml:27 (state 20) [(i = 0)]  
34: proc 1 (customer:1) bakery.Lock.pml:27 (state 21) [(i = 0)]  
35: spin: trail ends after 34 steps  
#Processes: 3  
34: proc 2 (customer:1) bakery.Lock.pml:31 (state 25)  
34: proc 1 (customer:1) bakery.Lock.pml:31 (state 25)  
34: proc 0 (init:1) bakery.Lock.pml:52 (state 3)  
3 processes created  
Exit Status 0

Queues

**3. Starvation free:**

The starvation free property was verified by using an assert in the promela code and by enforcing the weak fairness constraint.

Assert code:

```
assert(threadReachedCriticalSection[0] == eventually(threadReachedCriticalSection[1] == 3)
```

This is an assertion that will, for when the end state is reached, each of the two threads will be tested to ensure that they are equal to each other and 3.

Since the claim didn't throw an assertion error when it was executed, this means that each thread was eventually able to reach the critical section and update the variable to go from 0 to 3. Hence, the starvation free property isn't violated in the bakery lock implementation.

```
1 int MAX_NUM_THREADS = 2;
2 byte choosingThread[MAX_NUM_THREADS];
3 byte ticketNumber[MAX_NUM_THREADS];
4 byte threadReachedCriticalSection[MAX_NUM_THREADS];
5
6 proctype customer(){
7     byte id = _pid - 1;
8
9     int numIters;
10    for(numIters : 0 .. 2){
11        choosingThread[id]=1; // Threads request lock
12
13        int i, max=0;
14
15        for(i : 0 .. (MAX_NUM_THREADS - 1)){ // loop across all threads that are active.
16            // max is set to 0, if there is a ticketNumber that is greater than max, max = ticketNumber
17            if
18                :: ticketNumber[i] > max -> max = ticketNumber[i];
19            :: else;
20            fi;
21        }
22
23        // Take a ticket
24        ticketNumber[id] = max + 1;
25        choosingThread[id] = 0;
26
27        int j;
28        for(j : 0 .. (MAX_NUM_THREADS - 1)){
29            // Wait for our turn to come! |
30            // wait until it is our turn (choosingThread[j] == 1) threads turn to execute
31            do
32                :: (choosingThread[j] == 0) -> break;
33            od;
34
35            ((ticketNumber[j] == 0) || ((ticketNumber[j] >= ticketNumber[id]) && ((ticketNumber[j] != ticketNumber[id]
36            || (j >= id)))));
37        }
38        // This is the critical section of the code!
39        // It will do some - call a function/ update a data structure ETC
40        threadReachedCriticalSection[id]++;
41        // Release the lock by clearing our ticket in ticketNumber
42        ticketNumber[id] = 0;
43    }
44    printf("Process %d has completed %d iterations of this loop", id, numIters);
45 }
46
47 init {
48     run customer();
49     run customer();
50     (_nr_pr == 1);
51 }
52
```



Breandan Kerin 1431016 27th September 2018

# Advanced Computer Architecture

## Tutorial 1 - Question 2

Activities Lispin Thu 13:39

bakey\_lock.pml

Spin Version 6.4.6 -- 2 December 2016 :: Lispin Version 1.1.4 -- 27 November 2014

**Edit/View** **Simulate / Replay** **Verification** **Swarm Run** **<Help>** **Save Session** **Restore Session** **<Quit>**

Safety	Storage Mode	Search Mode
<input checked="" type="checkbox"/> safety <input checked="" type="checkbox"/> + invalid endstates (deadlock) <input checked="" type="checkbox"/> + assertion violations <input type="checkbox"/> + x/rxs assertions	<input type="checkbox"/> exhaustive <input type="checkbox"/> + minimized automata (slow) <input type="checkbox"/> + collapse compression <input type="checkbox"/> hash-compact <input type="checkbox"/> bistate/supertace	<input checked="" type="checkbox"/> depth-first search <input checked="" type="checkbox"/> + partial order reduction <input type="checkbox"/> + bounded context switching with bound: 0
<input type="checkbox"/> non-progress cycles <input type="checkbox"/> acceptance cycles <input type="checkbox"/> enforce weak fairness constraint	<input type="checkbox"/> do not use a never claim or tl property <input type="checkbox"/> use claim claim name (opt):	<input type="checkbox"/> + iterative search for short trail <input type="checkbox"/> breadth-first search <input checked="" type="checkbox"/> + partial order reduction <input checked="" type="checkbox"/> report unreachable code

**Show** **Advanced**

**Error** **Parameter**

**Trapping** **Settings**

**Options**

```

1  int MAX_THREADS = 2;
2  byte choosingThread[MAX_THREADS];
3  byte ticketNumber[MAX_THREADS];
4  byte threadReachedCriticalSection[MAX_THREADS];
5
6  proctype customer()
7  {
8      byte id = _pid - 1;
9
10     int numbers;
11     for(numbers : 0..2){
12         choosingThread[id]=1; // Threads request lock
13
14         int i, max=0;
15         for(i : 0..(MAX_THREADS - 1)){ // loop across all threads that are active.
16             // max is set to 0, if there is a ticketNumber that is greater than max, max = ticketNumber
17             if
18             :: ticketNumber[i] > max -> max = ticketNumber[i];
19             :: else;
20             fi;
21         }
22
23         // Take a ticket
24         ticketNumber[id] = max + 1;
25         choosingThread[id] = 0;
26
27         int i;
28         for(i : 0..(MAX_THREADS - 1)){
29             // Wait for our turn to come!
30             // if choosing[i] == 0, then we are on another thread and we must
31             // wait until it is our turn (choosingThread[i] == 1) threads turn to execute
32             do
33             :: (choosingThread[i] == 0) -> break;
34             od;
35
36             (ticketNumber[i] == 0) || (ticketNumber[i] >= ticketNumber[id]) && (ticketNumber[i] := ticketNumber[id] || (0 >= id));
37         }
38         // This is the critical section of the code!
  
```

spin -a -bakery\_lock.pml  
 spin: bakery\_lock.pml:37, Error: syntax error saw '}' = 125  
 spin: bakery\_lock.pml:34, Error: syntax error saw function-name: assert 'read' assert  
 gcc -DMEMLM=1024 -O2 -DXUSAFE -DSAFETY -DNOCLAIM -w -o pan pan.c  
 /pan -m10000  
 Ptd: 30551

(Spin Version 6.4.6 -- 2 December 2016)  
 + Partial Order Reduction

Full statespace search for:  
 never claim - (not selected)  
 assertion violations +  
 cycle checks - (disabled by -DSAFETY)  
 invalid end states +

hash conflicts: 1 (resolved)

Stats on memory usage (in Megabytes):  
 1.788 equivalent memory usage for states (stored (State-vector + overhead))  
 1.753 actual memory usage for states (compression: 98.03%)  
 state-vector as stored = 68 byte + 28 byte overhead  
 128.000 memory used for hash table (-w24)  
 0.534 memory used for DFS stack (-m10000)  
 130.194 total actual memory usage

unreached in procype customer  
 (0 of 45 states)  
 unreached in init  
 (0 of 4 states)

pan: elapsed time 0.01 seconds  
 No errors found -- did you verify all claims?

Breandan Kerin 1431016 27th September 2018  
Advanced Computer Architecture  
Tutorial 1 - Question 2

Activities | 1spin | Thu 13:40 | bakery.Lock.pml | Spin Version 6.4.6 -- 2 December 2016 :: 1spin Version 1.1.4 -- 27 November 2014

File Edit View Simulate / Replay Verification Swarm Run <Help> Save Session Restore Session <Quit>

Mode: Random, with seed: 123  
Interactive (for resolution of all nondeterminism)  
Guided, with trail: bakery.Lock.pml:trail browse  
Initial steps skipped: 0  
Maximum number of steps: 10000  
MSC max text width: 20  
MSC update delay: 25  
Track Data Values (this can be slow)

Output Filtering (reg. exps.)  
process ids:   
queue ids:   
var names:   
tracked variable:   
track scaling:   
(re)Run Stop Rewind Step Forward Step Backward

Background command executed:  
spin -p -s -r -X -v -n123 -l -g -k bakery.Lock.pml:trail -u10000 bakery.Lock.pml

Save in: msc.ps

```
1 int MAX_NUM_THREADS = 2;  
2 byte choosingThread[MAX_NUM_THREADS];  
3 byte ticketNumber[MAX_NUM_THREADS];  
4 byte threadReachedCriticalSection[MAX_NUM_THREADS];  
5  
6 procType customer()  
7  
8   byte id = _pid - 1;  
9  
10   int numbers;  
11   for(numbers : 0..2){  
12     choosingThread[id]=1; // Threads request lock  
13  
14     int l, max=0;  
15  
16     for(i : 0..(MAX_NUM_THREADS - 1)){ // loop across all threads that are active.  
17       // max is set to 0, if there is a ticketNumber that is greater than max, max = ticketNumber  
18       if  
19         :: ticketNumber[i] > max -> max = ticketNumber[i];  
20       fi;  
21     }
```

[Variable values, step 149]

```
MAX_NUM_THREADS = 2  
choosingThread[0] = 0  
choosingThread[1] = 0  
customer(1):i = 2  
customer(1):j = 2  
customer(1):max = 0  
customer(1):numIters = 3  
customer(2):i = 2  
customer(2):j = 2  
customer(2):max = 0  
customer(2):numIters = 3  
spin: bakery.Lock.pml:37, Error: syntax error saw '}'  
threadReachedCriticalSection[0] = 3  
threadReachedCriticalSection[1] = 3  
ticketNumber[0] = 0  
ticketNumber[1] = 0
```

```
138: proc 1 (customer:1) bakery.Lock.pml:37 (state 29) [(1)]  
139: proc 1 (customer:1) bakery.Lock.pml:28 (state 30) [(+1)]  
140: proc 1 (customer:1) bakery.Lock.pml:28 (state 23) [(i<=(MAX_NUM_THREADS-1))]  
141: proc 1 (customer:1) bakery.Lock.pml:33 (state 24) [(choosingThread[i]==0)]  
142: proc 1 (customer:1) bakery.Lock.pml:28 (state 30) [(+1)]  
143: proc 1 (customer:1) bakery.Lock.pml:37 (state 31) [false]  
144: proc 1 (customer:1) bakery.Lock.pml:40 (state 30) [threadReachedCriticalSection[id]=0]  
145: proc 1 (customer:1) bakery.Lock.pml:42 (state 37) [ticketNumber[id] = 0]  
146: proc 1 (customer:1) bakery.Lock.pml:43 (state 39) [false]  
147: Process 0 has completed 3 iterations of this loop;147: proc 1 (customer:1) bakery.Lock.pml:45 (state 44) [print("Process %d has completed  
148: %d iterations of this loop;1d,numIters)]  
149: proc 0 (init:1) bakery.Lock.pml:51 (state 3) [(n_p==1)]  
150: spin: trail ends after 149 steps  
#processes: 1  
149: proc 0 (init:1) bakery.Lock.pml:52 (state 4)  
3 processes created  
Exit>Status 0
```

Queues



**4. Liveness:**

Due to the fact that the bakery lock implementation is starvation free, the implementation is proven to uphold the liveness property.

This is because, for the implementation to have liveness, at least one thread must be able to enter the critical section continually. For a program to be starvation free, all threads present must be able to enter the critical section eventually.

From this, the starvation free property of the implementation proven above shows that the liveness of this implementation is true.

The code and screenshots for the liveness property are seen above in the starvation free section of this report.