## Question 1

a)[](req -> Xgrant U full)

b)<>(overflow U flush)

c)[](!deadlock)

d)[](sheldon\_goes\_to\_pennys\_door -> 3 knocks)

e)<>(jay\_gets\_married\_to\_megan\_fox)

## Question 2

a)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Run | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| No. of clients | 4 | 4 | 5 | 4 | 4 | 4 | 4 | 5 | 4 | 4 |

b)

It is possible for the museum to surpass its preset capacity due to lack of synchronization between the threads. Once the state is open, any number of clients can enter the museum as each client that enters does not have to check with the manager if the capacity will be exceeded upon them entering. So if there is only one spot left, two clients may enter by the time the manager does the next assertion to check if the maximum capacity is exceeded. By the time he does the check, the capacity will have been exceeded and the assertion will fail. It could be possible that the manager may have been sleeping. Since the default state is open, more than 3 clients (preset capacity) may have entered the museum and by the time the manager woke up to do the check, the capacity was exceeded. There are two possibilities why the preset capacity may have been exceeded.

c) assert ((!(state==wait))||((state==wait)&&(n<(max+2))));

d)

The simulation only fails once the number of clients exceeds the present capacity by 2 or more.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Run | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| No. of clients | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |

e) Comments are in the code.

f)

No errors

State-vector 56 byte, depth reached 9999, errors: 0

112012 states, stored

291469 states, matched

403481 transitions (= stored+matched)

0 atomic steps

hash conflicts: 3439 (resolved)

Stats on memory usage (in Megabytes):

7.691 equivalent memory usage for states (stored\*(State-vector + overhead))

5.758 actual memory usage for states (compression: 74.86%)

state-vector as stored = 38 byte + 16 byte overhead

64.000 memory used for hash table (-w24)

0.343 memory used for DFS stack (-m10000)

70.007 total actual memory usage

-The proctype client never ends so it never reaches the end state.

-The proctype manager does not reach the close or end states.

-Init never reaches the closed state.

unreached in proctype client

A2\_Q2.pml:37, state 22, "-end-"

(1 of 22 states)

unreached in proctype manager

A2\_Q2.pml:46, state 3, "state = closed"

A2\_Q2.pml:64, state 20, "-end-"

(2 of 20 states)

unreached in init

(0 of 7 states)

pan: elapsed time 0.99 seconds

pan: rate 113143.43 states/second

## Question 3

a)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Run | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Total Servings Taken | 25 | 5 | 4 | 42 | 10 | 5 | 21 | 5 | 5 | 9 |

b)

using statement merging

1: proc 3 (savage) A2\_Q3.pml:26 (state 4) [else] (Pot is not greater than 0 so savage does not take any food from it)

2: proc 2 (savage) A2\_Q3.pml:26 (state 4) [else]

3: proc 3 (savage) A2\_Q3.pml:26 (state 5) [(1)] (The savage skips to the start of the do-while loop)

4: proc 1 (savage) A2\_Q3.pml:26 (state 4) [else]

5: proc 2 (savage) A2\_Q3.pml:26 (state 5) [(1)]

6: proc 3 (savage) A2\_Q3.pml:26 (state 4) [else]

7: proc 2 (savage) A2\_Q3.pml:26 (state 4) [else]

8: proc 0 (cook) A2\_Q3.pml:12 (state 3) [((pot==0))] (Pot size is equal to 0 so it will have to be refilled)

9: proc 3 (savage) A2\_Q3.pml:26 (state 5) [(1)]

10: proc 2 (savage) A2\_Q3.pml:26 (state 5) [(1)]

11: proc 1 (savage) A2\_Q3.pml:26 (state 5) [(1)]

12: proc 3 (savage) A2\_Q3.pml:26 (state 4) [else]

13: proc 2 (savage) A2\_Q3.pml:26 (state 4) [else]

14: proc 3 (savage) A2\_Q3.pml:26 (state 5) [(1)]

15: proc 0 (cook) A2\_Q3.pml:12 (state 4) [pot = 4] (Pot size was equal to 0 so it is assigned 4 again)

16: proc 2 (savage) A2\_Q3.pml:26 (state 5) [(1)]

17: proc 3 (savage) A2\_Q3.pml:23 (state 1) [((pot>0))] (The pot size is greater than 0 so the savage represented by proc. 3 will help himself)

18: proc 3 (savage) A2\_Q3.pml:24 (state 2) [pot = (pot-1)] (The size of the pot gets decremented by one once a savage has taken food)

3 got food

19: proc 3 (savage) A2\_Q3.pml:25 (state 3) [printf('%d got food\\n',\_pid)] (Prints the process number as well as the process id of the savage that took the food out of the pot)

20: proc 3 (savage) A2\_Q3.pml:23 (state 1) [((pot>0))]

21: proc 3 (savage) A2\_Q3.pml:24 (state 2) [pot = (pot-1)]

3 got food

22: proc 3 (savage) A2\_Q3.pml:25 (state 3) [printf('%d got food\\n',\_pid)]

23: proc 3 (savage) A2\_Q3.pml:23 (state 1) [((pot>0))]

24: proc 3 (savage) A2\_Q3.pml:24 (state 2) [pot = (pot-1)]

3 got food

25: proc 3 (savage) A2\_Q3.pml:25 (state 3) [printf('%d got food\\n',\_pid)]

26: proc 3 (savage) A2\_Q3.pml:23 (state 1) [((pot>0))]

27: proc 3 (savage) A2\_Q3.pml:24 (state 2) [pot = (pot-1)]

28: proc 2 (savage) A2\_Q3.pml:26 (state 4) [else]

29: proc 1 (savage) A2\_Q3.pml:26 (state 4) [else]

30: proc 2 (savage) A2\_Q3.pml:26 (state 5) [(1)]

31: proc 0 (cook) A2\_Q3.pml:12 (state 3) [((pot==0))] (Pot size is equal to 0 so it will have to be refilled)

32: proc 1 (savage) A2\_Q3.pml:26 (state 5) [(1)]

33: proc 2 (savage) A2\_Q3.pml:26 (state 4) [else]

34: proc 1 (savage) A2\_Q3.pml:26 (state 4) [else]

35: proc 0 (cook) A2\_Q3.pml:12 (state 4) [pot = 4] (The size of the pot was equal to 0 so it is refilled)

3 got food

36: proc 3 (savage) A2\_Q3.pml:25 (state 3) [printf('%d got food\\n',\_pid)]

37: proc 2 (savage) A2\_Q3.pml:26 (state 5) [(1)]

38: proc 3 (savage) A2\_Q3.pml:23 (state 1) [((pot>0))]

39: proc 3 (savage) A2\_Q3.pml:24 (state 2) [pot = (pot-1)]

3 got food

40: proc 3 (savage) A2\_Q3.pml:25 (state 3) [printf('%d got food\\n',\_pid)]

41: proc 3 (savage) A2\_Q3.pml:23 (state 1) [((pot>0))]

42: proc 3 (savage) A2\_Q3.pml:24 (state 2) [pot = (pot-1)]

3 got food

43: proc 3 (savage) A2\_Q3.pml:25 (state 3) [printf('%d got food\\n',\_pid)]

44: proc 3 (savage) A2\_Q3.pml:23 (state 1) [((pot>0))]

45: proc 3 (savage) A2\_Q3.pml:24 (state 2) [pot = (pot-1)]

3 got food

46: proc 3 (savage) A2\_Q3.pml:25 (state 3) [printf('%d got food\\n',\_pid)]

47: proc 3 (savage) A2\_Q3.pml:23 (state 1) [((pot>0))]

48: proc 2 (savage) A2\_Q3.pml:23 (state 1) [((pot>0))]

49: proc 1 (savage) A2\_Q3.pml:26 (state 5) [(1)]

50: proc 3 (savage) A2\_Q3.pml:24 (state 2) [pot = (pot-1)]

3 got food

51: proc 3 (savage) A2\_Q3.pml:25 (state 3) [printf('%d got food\\n',\_pid)]

52: proc 3 (savage) A2\_Q3.pml:26 (state 4) [else]

53: proc 2 (savage) A2\_Q3.pml:24 (state 2) [pot = (pot-1)] (At this point 5 savages have taken from the pot since the last time it was refilled. Thus, the value of the pot size should be negative)

54: proc 3 (savage) A2\_Q3.pml:26 (state 5) [(1)]

2 got food

55: proc 2 (savage) A2\_Q3.pml:25 (state 3) [printf('%d got food\\n',\_pid)]

56: proc 3 (savage) A2\_Q3.pml:26 (state 4) [else] (Savages don’t take from the pot as the size has fallen below 0)

57: proc 2 (savage) A2\_Q3.pml:26 (state 4) [else]

58: proc 3 (savage) A2\_Q3.pml:26 (state 5) [(1)]

59: proc 1 (savage) A2\_Q3.pml:26 (state 4) [else]

60: proc 2 (savage) A2\_Q3.pml:26 (state 5) [(1)]

61: proc 3 (savage) A2\_Q3.pml:26 (state 4) [else]

62: proc 2 (savage) A2\_Q3.pml:26 (state 4) [else]

63: proc 0 (cook) A2\_Q3.pml:11 (state 1) [((pot<0))] (pot size went below zero)

64: proc 3 (savage) A2\_Q3.pml:26 (state 5) [(1)]

65: proc 2 (savage) A2\_Q3.pml:26 (state 5) [(1)]

66: proc 1 (savage) A2\_Q3.pml:26 (state 5) [(1)]

spin: A2\_Q3.pml:11, Error: assertion violated

spin: text of failed assertion: assert(0)

67: proc 0 (cook) A2\_Q3.pml:11 (state 2) [assert(0)] (assertion failed)

spin: trail ends after 67 steps

#processes: 4

pot = -1 (final pot size was -1)

67: proc 3 (savage) A2\_Q3.pml:20 (state 8)

67: proc 2 (savage) A2\_Q3.pml:20 (state 8)

67: proc 1 (savage) A2\_Q3.pml:20 (state 8)

67: proc 0 (cook) A2\_Q3.pml:8 (state 9)

4 processes created

c) Code can be found in the file A2\_Q3.pml.

d)

Means never claim has reached final state

**Output from running pan file:**

pan:1: end state in claim reached (at depth 45)

pan: wrote A2\_Q3.pml.trail

(Spin Version 6.2.2 -- 6 June 2012)

Warning: Search not completed

+ Partial Order Reduction

Full statespace search for:

never claim + (never\_0)

assertion violations + (if within scope of claim)

Bytes per state

acceptance cycles - (not selected)

invalid end states - (disabled by never claim)

State-vector 52 byte, depth reached 45, errors: 1

23 states, stored

Longest execution path

0 states, matched

23 transitions (= stored+matched)

0 atomic steps

hash conflicts: 0 (resolved)

Stats on memory usage (in Megabytes):

0.001 equivalent memory usage for states (stored\*(State-vector + overhead))

0.289 actual memory usage for states

64.000 memory used for hash table (-w24)

0.343 memory used for DFS stack (-m10000)

64.539 total actual memory usage

Total memory used for this formal verification run

pan: elapsed time 0 seconds

**Output from running trace file:**

starting claim 0

using statement merging

This expression is true which causes never claim to go to accept state

Never claim moves to line 12 [(!(((pot>0)&&(pot<4))))]

Starting round\_robin\_arbiter with pid 2

2: proc 0 (:init:) round\_robin.pml:73 (state 1) [(run round\_robin\_arbiter())]

Never claim moves to line 16 [(1)]

spin: trail ends after 3 steps

#processes: 2

pot = 4

req[0] = 0

req[1] = 0

req[2] = 0

req[3] = 0

gnt[0] = 0

gnt[1] = 0

gnt[2] = 0

gnt[3] = 0

3: proc 1 (round\_robin\_arbiter) round\_robin.pml:24 (state 16)

3: proc 0 (:init:) round\_robin.pml:76 (state 2)

3: proc - (never\_0) round\_robin.pml:17 (state 8) <valid end state>

2 processes created