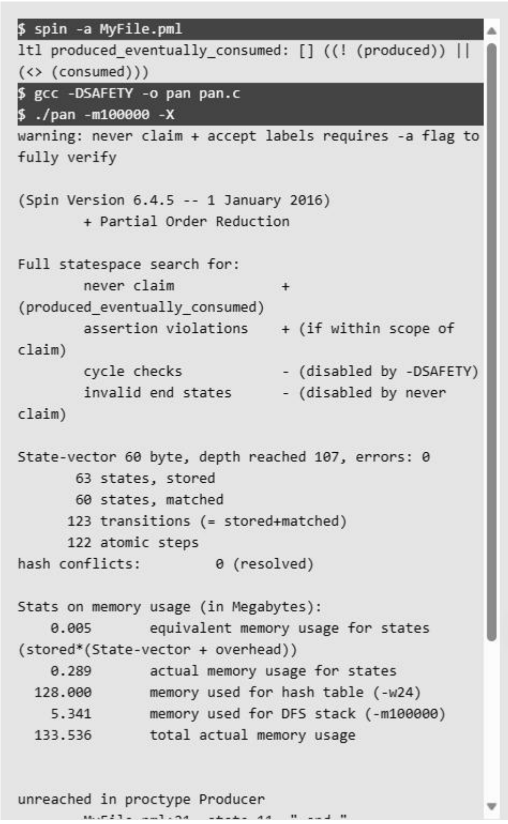
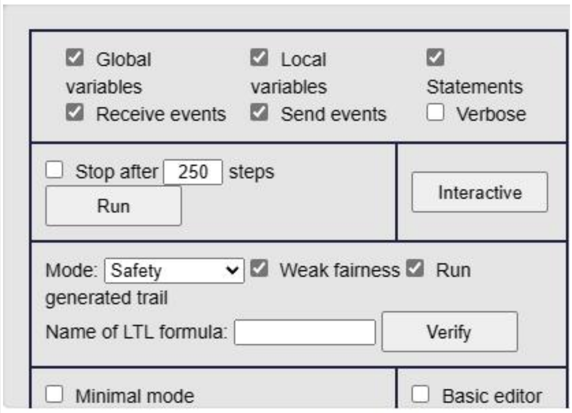
Q3.

Extended Model Explanation: The model was extended with two global variables, ***produced*** and ***consumed***, to track the production and consumption of values. In the Producer process, produced is set to true upon sending a value. Similarly, consumed is set to true in the Consumer process when a value is received.

LTL Property: The specified LTL property is ***[] ((! (produced)) || (<> (consumed)))***. This formula ensures that if a value is produced (indicated by produced being true), it will eventually be consumed (consumed becomes true).



```
$ spin -a MyFile.pml
ltl produced_eventually_consumed: [] ((! (produced)) ||
(<> (consumed)))
$ gcc -DSAFETY -o pan pan.c
$ ./pan -m100000 -X
warning: never claim + accept labels requires -a flag to
fully verify

(Spin Version 6.4.5 -- 1 January 2016)
        + Partial Order Reduction

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

State-vector 60 byte, depth reached 107, errors: 0
       63 states, stored
       60 states, matched
      123 transitions (= stored+matched)
      122 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.005       equivalent memory usage for states
(stored*(State-vector + overhead))
    0.289       actual memory usage for states
  128.000       memory used for hash table (-w24)
    5.341       memory used for DFS stack (-m100000)
  133.536       total actual memory usage


unreached in proctype Producer
```



Verification Output: The Spin model checker output shows no errors in the full state space search, indicating that the model satisfies the LTL property.

Conclusion: The absence of errors in the verification output confirms that in all possible states of the model, every produced value is eventually consumed.

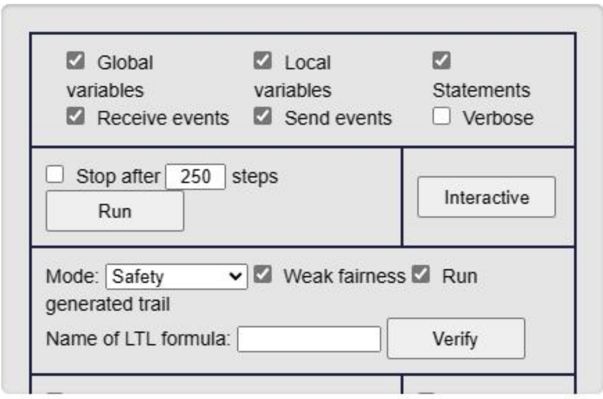
Q4.

Extended Model Explanation: The model has been extended to ensure that the same consumer does not consume twice in a row. This was achieved by introducing two variables: ***current_consumer*** and ***previous_consumer***. The current_consumer is set to the ID of the consumer process each time a consumer acts, and previous_consumer stores the ID of the last consumer who acted. After each consumption, current_consumer is reset to 0, and previous_consumer is updated to the current consumer's ID. This setup ensures that a consumer process can only act if it was not the last to do so, preventing consecutive actions by the same consumer.

LTL Property Explanation: The LTL ***(Linear Temporal Logic) property [] ((previous_consumer != current_consumer))*** is used to continuously ([]) check that previous_consumer and current_consumer are not the same.It ensures the same consumer does not consume twice in a row throughout the entire execution of the model.

Verification Output:

```
$ spin -a MyFile.pml
ltl not_the_same_consumer: []
((previous_consumer!=current_consumer))
$ gcc -DSAFETY -o pan pan.c
$ ./pan -m100000 -X
warning: never claim + accept labels requires -a
flag to fully verify

(Spin Version 6.4.5 -- 1 January 2016)
        + Partial Order Reduction

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

State-vector 60 byte, depth reached 56, errors: 0
       51 states, stored
       24 states, matched
       75 transitions (= stored+matched)
       74 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.004       equivalent memory usage for
states (stored*(State-vector + overhead))
    0.289       actual memory usage for states
  128.000       memory used for hash table (-w24)
    5.341       memory used for DFS stack (-
m100000)
  133.536       total actual memory usage
```



This output confirms that the verification was successful with no errors found, implying that the model satisfies the LTL property.

Clarity and Evaluation: The use of current_consumer and previous_consumer effectively prevents the consecutive action of the same consumer, and the LTL property continuously checks this condition, ensuring the correctness of the model's behavior in this regard.

Q5.

Extended Model Explanation: The model was extended to enforce fairness in consumption between two consumer processes. This was achieved by introducing a global array ***consumption_count[2]***, where each index corresponds to a consumer process and tracks the number of items it has consumed. The Consumer process updates this array each time it consumes an item

LTL Property Explanation: The LTL property fairness_in_consumption: ***[] ((((consumption_count[0]-consumption_count[1])<=1)) && (((consumption_count[1]-consumption_count[0])<=1)))*** was specified to ensure fairness in the consumption process. This property states that at all times, the difference in the number of items consumed by each consumer should not exceed 1. It's intended to prevent one consumer from dominating the consumption process.

Screenshot of the Verification Output:

```
$ spin -a MyFile.pml
ltl fairness_in_consumption: [] ((((consumption_count[0]-
consumption_count[1])<=1)) && (((consumption_count[1]-
consumption_count[0])<=1)))
$ gcc -DSAFETY -o pan pan.c
$ ./pan -m100000 -X
warning: never claim + accept labels requires -a flag to fully verify
pan:1: assertion violated  !( !((((consumption_count[0]-
consumption_count[1])<=1)&&((consumption_count[1]-
consumption_count[0])<=1)))) (at depth 12)
pan: wrote pmlfileb10MWH.trail
```

```
(Spin Version 6.4.5 -- 1 January 2016)
Warning: Search not completed
    + Partial Order Reduction
```

```
Full statespace search for:
    never claim             + (fairness_in_consumption)
    assertion violations    + (if within scope of claim)
    cycle checks        - (disabled by -DSAFETY)
    invalid end states  - (disabled by never claim)
```

```
State-vector 68 byte, depth reached 12, errors: 1
        5 states, stored
        0 states, matched
        5 transitions (= stored+matched)
        4 atomic steps
hash conflicts:         0 (resolved)
```

```
Stats on memory usage (in Megabytes):
    0.000   equivalent memory usage for states (stored*(State-vector +
overhead))
    0.289   actual memory usage for states
  128.000   memory used for hash table (-w24)
    5.341   memory used for DFS stack (-m100000)
  133.536   total actual memory usage
```

```
pan: elapsed time 0.01 seconds
$ spin -g -l -p -r -s  -t MyFile.pml
ltl fairness_in_consumption: [] ((((consumption_count[0]-
consumption_count[1])<=1)) && (((consumption_count[1]-
consumption_count[0])<=1)))
spin: warning, "MyFile.pml" is newer than MyFile.pml.trail
starting claim 2
using statement merging
Never claim moves to line 4 [(1)]
  2:    proc  1 (Producer:1) MyFile.pml:11 (state 1)    [((turn==P))]
  3:    proc  1 (Producer:1) MyFile.pml:11 Send 0   -> queue 1 (ch1)
  3:    proc  1 (Producer:1) MyFile.pml:11 (state 2)    [ch1!a]
        queue 1 (ch1): [0]
              The producer 1 -->sent 0!
  3:    proc  1 (Producer:1) MyFile.pml:12 (state 3)    [printf('The
producer %d -->sent %d!\\n',_pid,a)]
        queue 1 (ch1): [0]
  3:    proc  1 (Producer:1) MyFile.pml:13 (state 4)    [a = (1-a)]
        queue 1 (ch1): [0]
        Producer(1):a = 1
  3:    proc  1 (Producer:1) MyFile.pml:14 (state 5)    [turn = C]
        turn = C
```

```
        queue 1 (ch1): [0]
        Producer(1):a = 1
  5:    proc  3 (Consumer:1) MyFile.pml:25 (state 1)    [((turn==C))]
        queue 1 (ch1): [0]
  5:    proc  3 (Consumer:1) MyFile.pml:25 (state
2)    [current_consumer = _pid]
        queue 1 (ch1): [0]
        current_consumer = 3
  5:    proc  3 (Consumer:1) MyFile.pml:26 (state 3)    [index =
(_pid%2)]
        queue 1 (ch1): [0]
        current_consumer = 3
        Consumer(3):index = 1
  6:    proc  3 (Consumer:1) MyFile.pml:27 Recv 0   <- queue 1 (ch1)
  6:    proc  3 (Consumer:1) MyFile.pml:27 (state 4)    [ch1?b]
        queue 1 (ch1):
        Consumer(3):b = 0
  6:    proc  3 (Consumer:1) MyFile.pml:28 (state
5)    [consumption_count[index] = (consumption_count[index]+1)]
        queue 1 (ch1):
        consumption_count[0] = 0
        consumption_count[1] = 1
        Consumer(3):b = 0
                      The consumer 3 -->received 0!
```

```
  6:    proc  3 (Consumer:1) MyFile.pml:29 (state 6)    [printf('The
consumer %d -->received %d!\\n\\n',_pid,b)]
        queue 1 (ch1):
        consumption_count[0] = 0
        consumption_count[1] = 1
        Consumer(3):b = 0
  6:    proc  3 (Consumer:1) MyFile.pml:30 (state
7)    [assert((current_consumer==_pid))]
        queue 1 (ch1):
        consumption_count[0] = 0
        consumption_count[1] = 1
        Consumer(3):b = 0
  6:    proc  3 (Consumer:1) MyFile.pml:31 (state 8)    [turn = P]
        turn = P
        queue 1 (ch1):
        consumption_count[0] = 0
        consumption_count[1] = 1
        Consumer(3):b = 0
  8:    proc  1 (Producer:1) MyFile.pml:11 (state 1)    [((turn==P))]
        queue 1 (ch1):
  9:    proc  1 (Producer:1) MyFile.pml:11 Send 1   -> queue 1 (ch1)
  9:    proc  1 (Producer:1) MyFile.pml:11 (state 2)    [ch1!a]
        queue 1 (ch1): [1]
              The producer 1 -->sent 1!
```

```
  9:    proc  1 (Producer:1) MyFile.pml:12 (state 3)    [printf('The
producer %d -->sent %d!\\n',_pid,a)]
        queue 1 (ch1): [1]
  9:    proc  1 (Producer:1) MyFile.pml:13 (state 4)    [a = (1-a)]
        queue 1 (ch1): [1]
        Producer(1):a = 0
  9:    proc  1 (Producer:1) MyFile.pml:14 (state 5)    [turn = C]
        turn = C
        queue 1 (ch1): [1]
        Producer(1):a = 0
 11:    proc  3 (Consumer:1) MyFile.pml:25 (state 1)    [((turn==C))]
        queue 1 (ch1): [1]
 11:    proc  3 (Consumer:1) MyFile.pml:25 (state
2)    [current_consumer = _pid]
        queue 1 (ch1): [1]
 11:    proc  3 (Consumer:1) MyFile.pml:26 (state 3)    [index =
(_pid%2)]
        queue 1 (ch1): [1]
        Consumer(3):index = 1
 12:    proc  3 (Consumer:1) MyFile.pml:27 Recv 1   <- queue 1 (ch1)
 12:    proc  3 (Consumer:1) MyFile.pml:27 (state 4)    [ch1?b]
        queue 1 (ch1):
        Consumer(3):b = 1
 12:    proc  3 (Consumer:1) MyFile.pml:28 (state
5)    [consumption_count[index] = (consumption_count[index]+1)]
        queue 1 (ch1):
        consumption_count[0] = 0
        consumption_count[1] = 2
        Consumer(3):b = 1
                      The consumer 3 -->received 1!
```

```
 12:    proc  3 (Consumer:1) MyFile.pml:29 (state 6)    [printf('The
consumer %d -->received %d!\\n\\n',_pid,b)]
        queue 1 (ch1):
        consumption_count[0] = 0
        consumption_count[1] = 2
        Consumer(3):b = 1
 12:    proc  3 (Consumer:1) MyFile.pml:30 (state
7)    [assert((current_consumer==_pid))]
        queue 1 (ch1):
        consumption_count[0] = 0
        consumption_count[1] = 2
        Consumer(3):b = 1
 12:    proc  3 (Consumer:1) MyFile.pml:31 (state 8)    [turn = P]
        turn = P
        queue 1 (ch1):
        consumption_count[0] = 0
        consumption_count[1] = 2
        Consumer(3):b = 1
```

```
spin: _spin_nvr.tmp:3, Error: assertion violated
spin: text of failed assertion: assert(!(!((((consumption_count[0]-
consumption_count[1])<=1)&&((consumption_count[1]-
consumption_count[0])<=1)))))
Never claim moves to line 3 [assert(!(!((((consumption_count[0]-
consumption_count[1])<=1)&&((consumption_count[1]-
consumption_count[0])<=1)))))]
spin: trail ends after 13 steps
#processes: 4
        turn = P
        queue 1 (ch1):
        current_consumer = 3
        consumption_count[0] = 0
        consumption_count[1] = 2
 13:    proc  3 (Consumer:1) MyFile.pml:23 (state 10)
 13:    proc  2 (Consumer:1) MyFile.pml:23 (state 10)
 13:    proc  1 (Producer:1) MyFile.pml:9 (state 7)
 13:    proc  0 (Producer:1) MyFile.pml:9 (state 7)
 13:    proc  - (fairness_in_consumption:1) _spin_nvr.tmp:2 (state 6)
4 processes created
```

This output indicates an assertion violation, suggesting that the LTL property is not satisfied by the model.

Counterexample Explanation: The simulation revealed that the fairness property was violated when one consumer (***with _pid 3***) consumed twice before the other consumer had a chance to consume. Specifically, ***consumption_count[1]*** became ***2*** while ***consumption_count[0]*** remained ***0***, thereby violating the LTL property that requires the difference in consumption between consumers to be no more than 1.

Clarity and Evaluation: The counterexample found highlights a potential imbalance in the consumption process, where one consumer can consume more than once before the other has a chance, leading to a temporary unfair state. The goal is to achieve a balance between ensuring fairness and reflecting the realistic behavior of the system.