Danmarks Tekniske Universitet

02158 Concurrent Programming

Mandatory Assignment 2 Resubmission - Group 12

Søren Sehested Schubye
s224756

Lucas Juel Sjøstrøm
s224742

11. november 2024

Indholdsfortegnelse

1 Problem 1 - Resubmission

For the resubmission we have changed the code as suggested resetting the variable i.

We have created a fair solution to the lab problem by introducing a lock mechanism. This lock is set as true for every iteration of the do-loop in the coordinator. The coordinator can then only set the ok-flag if both the lock and enter[i] is true, and incrit == 0.

```
bool lock = true;
  int i = 0;
  do
  ::
   if
   ::i < N ->
    if
    :: enter[i] && incrit == 0 && lock;
     ok[i] = true;
     lock = false;
     ok[i] == false ->
    :: else ->
     skip;
    fi;
    i++;
    lock = true
   :: i >= N -> break;
   fi
  od
```

Also when this step is entered, we wait for the exit-stage to set the ok-flag to false. We skip the iteration if the if-statement is not met, and then we add 1 to the counter variable i, and set the lock back to true, so another process is able to go into the critical section.

Every process is looked at once for each round in the Coordinators do-loop. This means, all processes have equal chance of being accepted into the critical section.

Also a check in jSpin with the ltl fomular:

```
ltl fairness { []((P[0]@entry) -> <>(P[0]@crit)) }
```

further ensures fairness.

We can thus conclude, that this solution achieves fairness and satisfies mutual exclusion.

2 Problem 2

In order to make sure two instances of a car does not try to move to the same tile, each tile must be initialized with a semaphore containing 1 permit. Such that when a car enters a tile, it calls field.enter() and takes the ticket thus preventing other cars from entering. The other cars will, because of the Semaphore wait, until a ticket reappears. When the car then leaves it calls field.leave() and puts the ticket back.

3 Problem 3 - resubmission

3.1 MonoAlley Solution

The monoAlley implements the use of a semaphore inorder to ensure mutual exclusion. When a car enters the alley, the P operation is called ”taking a coconut”if any are available else it will wait.

When a car leaves the alley, it calls the V operation, ”putting back the coconut”notifying a car and let’s it enter the alley.

3.2 Alley Safety Property

The alley safety property, is the property that ensures that all cars in the alley is going the same direction. We wanted to implement the jSpin code, much like the code in Problem 1 was implemented, meaning a ”crit”stage was at some point met.

```
if
:: _pid < 5 ->
    assert(!(in_critical[5] || in_critical[6] ||
    in_critical[7] || in_critical[8]));
else ->
    assert(!(in_critical[0] || in_critical[1] ||
    in_critical[2] || in_critical[3] || in_critical[4]));
fi
```

Other than trying to follow the java code as much as possible, we implemented some helper lists, that has boolean values for each car, wether they’re in the critical zone or not.

When they then enter the critical zone, the in_critical flag is set to true, and an assertion is made. The assertion wether both the 5th and 6th car is in the critical section. If they are, the assertion fails.

4 Problem 5 - resubmission

4.1 Fixing the MultiAlley solution

In the MultiAlley implementation it relied on the first car grabbing the semaphore for cars going the opposite direction in order to block them. However since nothing stops the cars from calling the enter function concurrently, this created race conditions, where cars in both directions could acquire their respective semaphores and then become stuck since they are blocking for each other. This could be fixed, by adding a third semaphore. A shared lock semaphore that shuts access to the cars going in the other direction, which will then be released when the last car in one direction have left the alley.

In the modified man2safe promela model the non atomicity of the counters are kept, but assertions have been inserted to test them. The upSem and downSem should however protect the counters. Through verification in jspin, it can be concluded that the alley safety property is met. Meaning that no deadlocks occur and only cars going in the same direction enters the alley at the same time.

4.2 Starvation

The modified implementation however does not address the one direction bias that the cars might experience. Meaning, once a car enters the alley, the other cars going in the same direction might be perfectly spread out such, that a car going that direction will always be in the alley. This will cause starvation for the cars going the other direction, since the lock is never returned.

Since it is only required by the first car going down the alley to take both the downSem and lock semaphores, it will favor continuous flow in the alley, since the subsequent cars only have to pick up one semaphore.

With regards to slowing down the speed, this will only increase starvation, since it will make it more likely for cars going in the same direction to accumulate in the alley, thus preventing cars going the other direction from entering.

5 Conclusion

We can conclude, that it is not always beneficial to use semaphores over busy wait. In systems where longer wait times are expected, semaphores has a huge advantage, because of the wait/-notify mechanism. And because the thread is then waiting, it is not wasting resources. On the contrary if the wait is short, the upside of the thread sleeping becomes a downside, as you loose more efficiency due to the use of context switching.

Using a tool like jSpin to analyze and test synchronization, has both advantages and disadvantages. If we overlook the time it takes to learn how to use jSpin, the interface is also quite outdated and the output from the "Guided-runs were mostly unreadable. We usually aquired the help of chatGPT or other generative AI models to format the output of jSpin. Yet the upside of having it give error messages, the use of assertion steps and LTL are really useful in order ensure liveness properties and satisfy mutual exclusion.