Homework 1

Hanlin He (hxh160630)

September 12, 2018

1 Exercise 11

1.1

The protocol satisfies mutual exclusion.

Proof. Prove by contradiction.

Let $W_X(var := val)$ denote a write operation by X assigning value val to variable var, and $R_X(var = val)$ denote a read operation by X on variable var and get a value val.

Let process A is able to enter critical zone after some iterations, then there should be time line like this:

$$W_A(turn := A) \to R_A(busy = false) \to W_A(busy := true) \to R_A(turn = A)$$

Assume a process B is able to enter critical zone too, then B need to go through a similar timeline.

$$W_B(turn := B) \to R_B(busy = false) \to W_B(busy := true) \to R_B(turn = B)$$

For $R_A(turn = A)$ to be true, $W_B(turn := B)$ can not happen between $W_A(turn := A) \to \ldots \to R_A(turn = A)$. Otherwise, $R_A(turn = A)$ would fail. Thus we can determine $W_B(turn := B)$ can only happen either before $W_A(turn := A)$ or after $R_A(turn = A)$.

- If $W_B(turn := B)$ happen before $W_A(turn := A)$, for B to be able to enter critical zone, $R_B(turn = B)$ must happen before $W_A(turn := A)$, otherwise $R_B(turn = B)$ would fail. Therefore we have the following happen before relations:

  $$W_B(turn := B) \to R_B(turn = B) \to W_A(turn := A) \to R_A(turn = A)$$

  which would lead to the following contradiction:

  $$R_B(busy = false) \to W_B(busy := true) \to R_A(busy = false)$$

- Likewise, if $W_B(turn := B)$ happen after $R_A(turn = A)$, would leed to the following contradiction:

$$R_A(busy = false) \rightarrow W_A(busy := true) \rightarrow R_B(busy = false)$$

In conclusion, given process A could enter critical section, process B will never be able to enter critical section. The protocol satisfies mutual exclusion. $\square$

1.2

The protocol is not starvation free. From section 1.1 we can see for a process A to enter critical section, no other process can perform a write operation on *turn* between these operations.

$$W_A(turn := A) \rightarrow R_A(busy = false) \rightarrow W_A(busy := true) \rightarrow R_A(turn = A)$$

So it is possible for a process A failed to perform $R_A(turn = A)$ in every iteration, since *turn* could possibly be overwritten by other process every time before $R_A(turn = A)$ and after $W_A(turn := A)$, causing A to starve.

1.3

The protocol is not deadlock free. Consider the following interleaving for two process A and B.

$$W_A(turn := A)$$
$$\downarrow$$
$$W_B(turn := B)$$
$$\downarrow$$
$$R_A(busy = false)$$
$$\downarrow$$
$$W_A(busy := true)$$
$$\downarrow$$
$$R_B(busy)$$
$$\downarrow$$
$$R_A(turn)$$

Since $W_A(busy := true)$ happen before $R_B(busy)$, process B will keep spinning on `while(busy)`. On the other hand, $R_A(turn)$ happen after $W_B(turn := B)$, so `while(turn != me)` would fail for process A. A would start another iteration and would get stuck in spinning on `while(busy)` since $busy = true$. And from this moment, all processes attempting to acquire lock would get stuck.

2 Exercise 12

15

`Filter` lock spins on $\texttt{while}((\exists k \neq me)(level[k] \geq i \wedge victim[i] = me))$. During the interval a process p checking this condition, other processes could perform the same check and passed, since $level[p] \geq i$ would be true. So it is possible for other processes overtake a 'slower' process arbitrary times during the interval the 'slower' process check the condition.

Incomplete proof. Some process sets itself victim and cannot overtake p.

3 Exercise 14

25

There are two changes needs to be made to adjust the `Filter` lock to support l-mutual exclusion.

First, changing the for loop condition in `lock()` to

```
for(int i=1; i <= n-l; i++)
```

This could reduce the level to get to critical section. Basically, the `Filter` allow in total l processes for whose $level[p] > n - l$. So if we let process enter critical section if it reached level $n - l$, we are actually allowing l process enter critical section.

Then, we also need to change the spinning condition to

$$\texttt{exists at least n-l } k \texttt{ where } level[k] \geq i$$

so that there are at most l processes in higher levels instead of only one. This will ensure the processes will get to level $n - l$ if there are less than l processes in critical section.

4 Exercise 15

25

The wrapper does not satisfy mutual exclusion. Consider the following interleaving:

Table 1: Possible Interleaving

Process A	Process B	Step
$i_A = Index_A$		(1)
	$i_B = Index_B$	(2)
$x = i_A$		(3)
	$x = i_B$	(4)
$\texttt{while}(y \neq -1)\{\}$		(5)
	$\texttt{while}(y \neq -1)\{\}$	(6)
$y = i_A$		(7)
	$y = i_B$	(8)
$\texttt{if}(x \neq i_A)$		(9)
	$\texttt{if}(x \neq i_B)$	(10)

After step (8), $x = i_b$, so process B will get the lock right away. Meanwhile, process A would go into line `lock.lock()`, where it takes the 'long path' to get the lock. There is actually no other process attempting to acquire the internal `lock` object, since process B get the lock without acquiring the internal lock. So process A will directly get the internal `lock` and return. Two processes enter critical section at the same time.

Thus the wrapper does not satisfy mutual exclusion.