Styresystemer og Multiprogrammering

G-opgave #3

Jenny-Margrethe Vej
(`rwj935@alumni.ku.dk`)
Klaes Bo Rasmussen
(`twb822@alumni.ku.dk`)

Datalogisk Institut, Københavns Universitet
Blok 3 - 2013

1 Locks and condition variables for kernel threads in Buenos

1. First task should be completed, it goes as follows:

`lock_reset`
This resets the spinlock and the `locked` variable in the given `lock`.

`lock_acquire`
Saves the interrupt status, setting it to disabled. Then acquires spinlock for the given lock. It runs and keeps checking if the lock is still locked. If so, it adds the lock to the sleep queue, releases the spinlock and switches thread. When it is no longer locked, it locks it again so its locked to itself and the code can run without disruptions.

`lock_release`
Disables interrupts, saving the status. Then acquires the spinlock. If lock is no longer locked, wake it up and continue to release spinlock and set the interrupt status back to its former glory.

2. We don't know how to test this, but at least we get what this was supposed to do now.

`condition_init(cond_t *cond)`
This doesn't need to do anything.

`condition_wait(cond_t *cond, lock_t *lock)`
Since it is required that this is only called when the lock is already acquired, we need to release it first. Then add the condition to the sleepq. Switch to a new thread until we return back here, and then reacquire the lock so it can be released when we get out of the `condition_wait`.

`condition_signal(cond_t *cond, lock_t *lock)`
`lock` is set to itself, to satisfy the compiler. Then we signal, or rather wake up the sleeping condition.

`condition_broadcast(cond_t *cond, lock_t *lock)`
Same as with the `condition_signal`, it just wakes up all sleepers.

2 One-Lane Synchronisation

1. When syncronising a stretch of road like this, it would be preferrable to account for different times of day when traffic amount changes. But if we consider a simple solution:

Let the first car that gets to the crossing be put into the queue first, let it pass. If no other cars get to the crossing while the first is passing it, reset the queue when the car exits.

If more cars come, open up for new cars going in the same direction. When a car approaches from the other direction, set a timer of 30 seconds before cars need to stop from the first direction, let the rest through before opening up for the other direction, given the same rules. Again, if there are no cars as the last crossing car leaves the narrow path, reset the queue.

2. We did not manage to complete this part of the task yet. But we did make an attempt at it. It was hard to figure out exactly how to code it, but we will attempt to clarify what we did make.

First in the oneLane.h file, we got definitions for some directions, north, south and no direction at all. Then we create the mutexes and the cond_t's that we need to do the task. Some functions that we first considered necessary to complete the task are also there. These are specified in:

`oneLane.c`

We define some counters, for approaching and departing busses in each direction. Also which direction is currently waiting for the other side.

`stop_other_direction()`

This says, that if we are waiting for some side at all, attempt to stop busses from that side, to open up for busses from the direction we want. It's a conditional pthread which is used, to wait while there are still cars departing from the narrow path in the other direction. While it does this, it goes to sleep and waits using pthreads. When there are no more cars, we can unlock again and do whatever we need to do when one side has stopped running. Same goes for the other direction, with roles reversed. So far this doesn't really do much, but we are supposed to tell the approaching cars somehow to stop coming in from whatever direction the other direction is, so the counter of departing busses can go to zero.

The `go_this_way` function is quite uselss as long as we can already stop the other direction, it is pretty much a different word for the same thing. It would be more useful if we had to direct the traffic in some certain way, that an end user would be able to start the flow of busses in a direction he saw struggling to keep up. But for an automated solution this is in fact void of usability.

3. Couldn't finish this part without part two.