Algorithms Design for Bridge Problem

Hanlin He[*], Lizhong Zhang[†]

October 19, 2018

Semaphores Algorithm

The solution using semaphores is shown in algorithm 1.

Algorithm 1 Attempt Semaphores Solution

int[2] $count \leftarrow \{0, 0\}$
semaphore[2] $mutex \leftarrow \{1, 1\}$
semaphore $service \leftarrow 1$
semaphore $resource \leftarrow 1$

function arriveBridge($direction$) ▷ Assume $direction$ is 0/1
    down($service$)
    down($mutex[direction]$)
    $count[direction] \leftarrow count[direction] + 1$
    if $count[direction] = 1$ **then** ▷ First one need to acquire $resource$
        down($resource$)
    end if
    up($mutex[direction]$)
    up($service$)
end function

function leaveBridge($direction$)
    down($mutex[direction]$)
    $count[direction] \leftarrow count[direction] - 1$
    if $count[direction] = 0$ **then** ▷ Last one need to release $resource$
        up(resource)
    end if
    up($mutex[direction]$)
end function

[*]hxh160630@utdallas.edu
[†]lxz160730@utdallas.edu

Analysis

This algorithm mimic the reader logic in the 'Fair Readers-Writers Solution'. It uses array of integers/semaphores for two directions and direction as the reference index into the array. Each $mutex[i]$ ensure exclusive read/write operation on each $count[i]$. Traffics from one direction can enter the bridge without waiting if no traffic was waiting on the other side (just like any number of readers can read if no writer was waiting). And as long as there is some traffics waiting on the other side, the after traffic will wait until the other end traffic enter bridge once.

But this algorithm relies on the implementation of semaphores to ensure that all cars will come one direction after the other if both directions have traffic waiting. Consider scenario as follow:

1. Process A first called `arriveBridge` from direction 0.
2. Before A return,
   (a) Process B called `arriveBridge` from direction 1.
   (b) Process C called `arriveBridge` from direction 0.
3. Process A return from `arriveBridge`.

By requirement, the next process 'cross the bridge' should be B. In the algorithm, both B and C are 'spinning' on `down(service)`. After A invoke `leaveBridge` and release $service$, it depends on the implementation of the semaphore to determine which process among B and C could acquire the $service$. It's possible one side traffic got overtaken by the other side any number of times, but eventually the traffic will leave the bridge.

Monitors Algorithm

The solution using monitors is shown in algorithm 2.

Algorithm 2 Attempt Monitors Solution

```
int active[2] ← {0, 0}
int waiting[2] ← {0, 0}
condition canPass[2]

function arriveBridge(direction)                    ▷ Assume direction is 0/1
    other ← 1 − direction
    if active[other] + waiting[other] > 0 then
        waiting[direction] ← waiting[direction] + 1
        wait(canPass[direction])
        waiting[direction] ← waiting[direction] − 1
    end if
    active[direction] ← active[direction] + 1
end function

function leaveBridge(direction)
    other ← 1 − direction
    active[direction] ← active[direction] − 1
    if active[direction] = 0 ∧ waiting[other] > 0 then
        if waiting[direction] > 0 then
            signal(canPass[other])
        else
            signalAll(canPass[other])
        end if
    end if
end function
```

Analysis

This algorithm mimic the reader logic in the 'Starvation Free Readers-Writers Solution with Monitors'.

During `leaveBridge`, if current car is the last of its direction to leave the bridge for the moment, and there are cars waiting on the other side, prepare to signal the other end to pass. And if current car's direction also has cars waiting, then only signal one car from the other end to enter, otherwise, signal all cars on the other direction to enter.