Ask me anything

Mentimeter

2 questions
0 upvotes

Which is more fair — an intersection with traffic lights or a roundabout? What are the tradeoffs?

Mentimeter

An intersection with traffic lights is more fair, as in a roundabout starvation may occur (priority from the left)

traffic lights can be more fair, if done right. Roundabouts can be unfair if from one direction many cars are coming - then some have to wait for ages

I'd say technically traffic lights as you're guaranteed to eventually be admitted.

The intersection with traffic lights are fairer but at the cost of resource waste

Traffic lights guarantee fairness, at roundabouts you could theoretically experience starvation.

traffic lights is more fair. In a roundabout a livelock could occur if a car wouldn't leave the roundabout

Intersection (if programmed sanely :P) can guarantee that any participant will get to move within at most n cycles. With a roundabout it's up to each participant to get into it.

Traffic lights -> no starvation possible

The intersection, since everyone can eventually cross but we loose some traffic efficiency

12

Which is more fair — an intersection with traffic lights or a roundabout? What are the tradeoffs?

Mentimeter

We might have some starvation with roundabouts (e.g. a set of cars blocking the roundabout). Lights are fairer

An intersection with traffic lights as you can be sure that a certain moment and time you will get through, as with the roundabout if cars keep coming from your left, you will have to always let them pass.

Weak fairness allows a process to starve.

12

A roundabout can be unfair to East-West traffic if there is a steady flow of North-South traffic. How could you make it more fair?

Mentimeter

Adding an underpass

Add a rule to give priority for long waiting cars (would not work in practice i think :-))

Add smart camera that can detect heavy traffic.

This is already implemented in Bern.

4

In what way is strong fairness "stronger" than "weak fairness"?

Mentimeter

we don't need to continuously try to get eventually in

More equal

For strong fairness it is sufficient to make infinitely many requests, but they needn't be continuous

In weak fairness, some of the processes might starve while in strong fairness nobody will be left out

Strong fairness in the sense that you don't have to continually wait for the resource to be available.

Weak fairness allows a process to starve.

Weak fairness may allow a process to starve, it is useful when there aren't many processes, strong fairness will act more aggressively so it will be sure that no one will starve even if it interrupts other processes.

The guarantees made in 'strong' fairness are stronger

Stronger because you serve the resources with a FCFS principle.

9

In the Readers and Writers example code, why are the doRead and doWrite methods unsynchronized?

Mentimeter

```
public abstract class ReadersWritersStateTracking {
    ...
    public void read() {
        beforeRead();
        doRead();
        afterRead();
    }
    ...
}


class ReadWriteDemo extends ReadersWritersStateTracking {
    ...
    protected void doRead() {
        System.out.print("(");
        Thread.yield();
        System.out.print(")");
    }
    ...
}
```

Synchronization is ensured by the before / after methods.

Because the beforeRead and the afterRead are synchronized.

It does not need synchronization, beforeRead assures that no writer may enter during doRead.

doRead and afterRead are already synchronized, e.g. aquiring a lock. DoRead() is then the cirtical section code which is "Protected"

because the before methods make sure we can safely do the action (synced state variables)

because synchronization is managed in the before and after read/write

They do not need to be synchronized, because beforeRead / beforeWrite, take care that all conditions are met, when the thread reaches doRead()

Because doRead does not change the state of the object and many readers can perform read while there are not writers there

You can't come to doRead() if beforeRead() is not done, which is synchronised. Therefore doRead() is always in a valid stat

9

What liveness problems can arise if an object with many methods each acccessing only some fields has separate locks for each field?

Mentimeter

Deadlock

The order in which we aquire this locks is not trivial

Overhead if a method needs several fields

wait-for cycles by its methods on its fields, leading to deadlocks

deadlock

Deadlocks could be an issue, if the locks are not released while waiting.

Deadlock, if methods need multiple locks and the other conditions for a deadlock hold.

some threads could starve

We will have some processes that may end up deadlocked at a point in time when the chain builds up?

9

Mentimeter

Last chance for questions