Parallel Programming
Exercise Session 9

Outline

1. Feedback: Assignment 8
2. Assignment 9

Feedback: Assignment 8

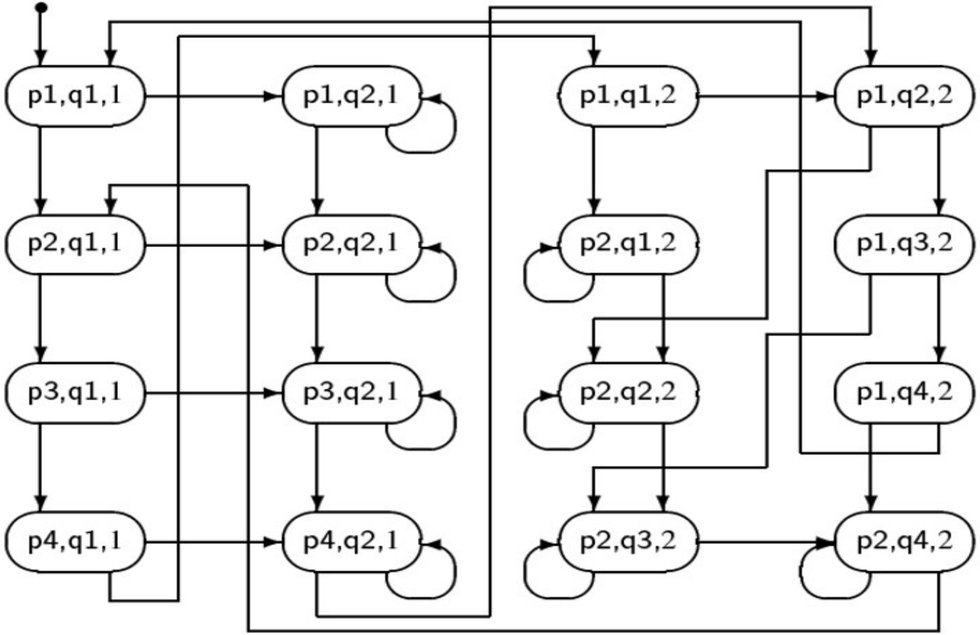
Recap: Critical Section Properties

- **Mutual exclusion**: No more then one process executing in the critical section
- **Progress**: When no process is in the critical section, any process that requests entry must be permitted without delay
- **No starvation (bounded wait)**: If any process tries to enter its critical section then that process must eventually succeed.

P	
p1:	Non-critical section P
p2:	while turn != 1
p3:	Critical section
p4:	turn = 2

turn = 1

Q	
q1:	Non-critical section Q
q2:	while turn != 2
q3:	Critical section
q4:	turn = 1



- **Mutual exclusion**: E.g. State (p3,q3,_) is not reachable
- **Progress**: E.g. There exists a path for P such that state (P3, _ , _) is reachable from (P2,_,_). Typical couterexamples: deadlocks and livelocks
- **No starvation (bounded wait)**: Possible starvation reveals itself as cycles in the state diagram.

P

p1: Non-critical section P
p2: while turn != 1
p3: Critical section
p4: turn = 2

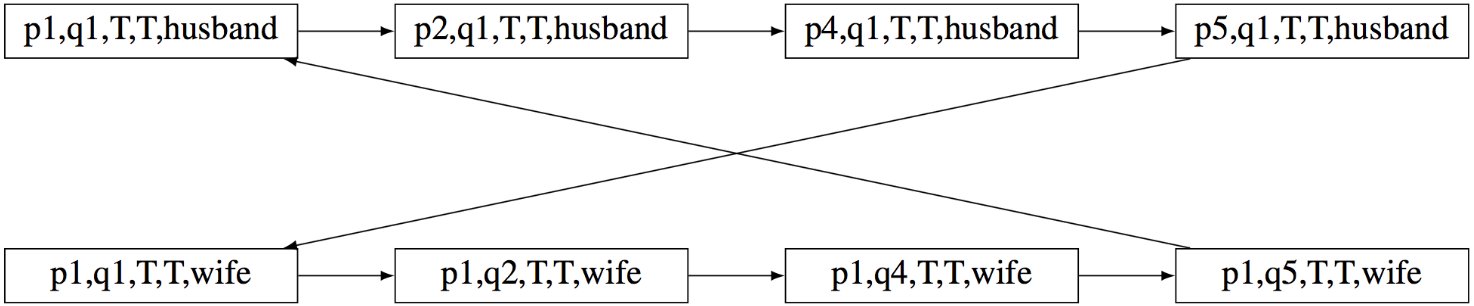
turn = 1

Q

q1: Non-critical section Q
q2: while turn != 2
q3: Critical section
q4: turn = 1

Feedback for Assignment 8

owner			
husband.hungry = true			
wife.hungry = true			
husband		wife	
p1:	while hungry	q1:	while hungry
p2:	owner != me	q2:	owner != me
p3:	sleep	q3:	sleep
p4:	spouse == hungry	q4:	spouse == hungry
p5:	owner = spouse	q5:	owner = spouse
p6:	CR	q6:	CR
p7:	hungry = false	q7:	hungry = false
p8:	owner = spouse	q8:	owner = spouse



Feedback for Assignment 8

- One way to solve the livelock problem is to impose an ordering when acquiring the lock on the shared resource.
- Or one of the spouses can actually take the spoon after certain number of retries

Feedback for Assignment 8

Optimistic vs Pessimistic concurrency control

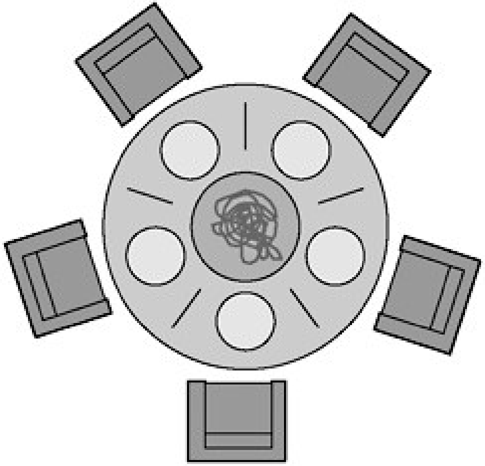
```java
@Override
public int nextInt() {
    // get the current seed value
    long next;
    synchronized (this) {
        long orig = state;
        // using recurrence equation to generate next
        next = (a * orig + c) & (~0L >>> 16);
        // store the updated seed
        state = next;
    }
    return (int) (next >>> 16);
}
```

```java
@Override
public int nextInt() {
    while (true) {
        // get the current seed value
        long orig = state.get();
        // using recurrence equation to generate next seed
        long next = (a * orig + c) & (~0L >>> 16);
        // store the updated seed
        if (state.compareAndSet(orig, next)) {
            return (int) (next >>> 16);
        }else{
            try {
                Thread.sleep(1);
            } catch (InterruptedException e) {

            }
        }
    }
}
```

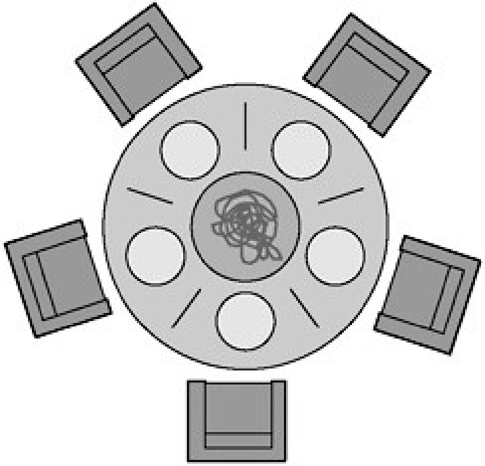
Assignment 9

Task 1 - Dining Philosophers



Originally proposed by E. W. Dijkstra
Imagine five philosophers who spend their lives thinking and eating.
They sit around a circular table with five chairs with a big plate of spaghetti.
However, there are only five chopsticks available.

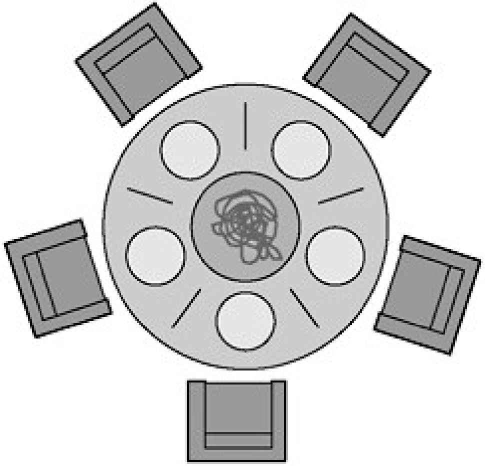
Task 1 - Dining Philosophers



Each philosopher thinks and when he gets hungry picks up the two chopsticks closest to him.

- If a philosopher can pick up BOTH chopsticks, he eats for a while.
- After a philosopher finishes eating, he puts down the chopsticks and starts to think again.

Find a solution that...



- Makes deadlocks impossible
- Has no starvation
- More than one parallel eating philosopher is possible

Task 2: Reasoning about Locks

You know two ways how to prove the correctness of a lock from the lecture

- State-space diagram

- Proof by contradiction (both in CS)

apply one of them in this task!

Task 3: JMM Basics – Transitive Closure

- Relation:
  Two sets, X and Y, a relation is a set of ordered pairs (x,y) such that x in X and y in Y.

Special case: X=Y

Example: - the “greater than” relation on natural numbers
- Statement 1 always directly follows Statement 2

Task 3: JMM Basics – Transitive Closure

Transitive closure of a relation R:

The smallest relation R’ such that:
If (a,b) is in relation R then (a,b) is in R’
If (c,d) and (e,f) are in R’ then (c,f) is in R’