THE UNIVERSITY OF SYDNEY

SOFT3410 Lab 3

Livelock and Starvation

The goal of this lab is to illustrate the problem of livelock and starvation. By the end you will have seen examples of livelock, and be able to understand how the misuse of locks can generate livelock.

Exercise 1: Safely transferring money between accounts

In last week's tutorial we successfully protected transfer of money between two accounts, but also sometimes prevented the transfer, and all subsequent transfers, entirely.

The problem arises because two locks must be held to transfer funds, but they are acquired sequentially. Lock A may be acquired by thread 1 and Lock B by thread 2 simultaneously. If thread 1 now tries to acquire lock B then it must wait for thread 2. Thread 2 is waiting for lock A, held by thread 1. Thus, deadlock occurs.

As we saw last week, if the locks are acquired in the same order then we can avoid deadlock. Modify your code from last week so that one lock is always acquired before the other. To do this you will need to compare the two accounts to determine an ordering before the locks are acquired.

Will your solution still be reliable if we add more accounts? Can the ordering you chose ever change, given the same locks?

Duration: 20 min

Exercise 2: Livelock

By carefully ordering the acquisition of locks we can avoid deadlock. However, we still need to wait for a lock to be freed. With multiple accounts we will often have the situation where we have locked one account, but are still waiting for the second account. Other transfers may then be waiting for the first lock to be freed. We know that eventually all transfers will be processed, but it would be more efficient if there was less waiting involved.

```java
public class TransferMoney implements Runnable {
    private Bank from;
    private Bank to;
    public TransferMoney(Bank from, Bank to) {
        this.from = from;
        this.to = to;
    }
    public void run() {
        for (int i = 0; i < 60; ++i) {
            try {
                Thread.sleep(50);
            } catch (Exception e) {
                System.err.println("Already interrupted.");
            }
            from.lock.lock();
            System.out.println("Acquired first lock on (" + from.hashCode() + ")");
            System.out.println("Attempting to acquire second lock on (" + to.hashCode() + ")"
            doImportantStuff();
            // Try to acquire second lock
            // If it's not free, then release the first lock while we wait.
            while (!to.lock.tryLock()) {
                System.out.println("Could not acquire second lock on (" + to.hashCode() + ")"
                from.lock.unlock();
                System.out.println("Released lock on (" + from.hashCode() + ") and starting ov
                from.lock.lock();
                System.out.println("Acquired first lock on (" + from.hashCode() + ")");
                System.out.println("Attempting to acquire second lock on (" + to.hashCode() +
                doImportantStuff();
            }
            // subtract money from one
            // add money to the other
            // release locks
        }
    }
    // A method to represent the extra work performed in a more complex system.
    protected void doImportantStuff() {
        try {
            Thread.sleep(50);
        } catch (Exception e) {
            System.err.println("Already interrupted.");
        }
    }
}
```

In this code, if the second lock is not available we temporarily release the first lock then try again. Note that this method does not just (potentially) reduce waiting time, it also means that we have not had to determine a safe ordering of lock acquisition.

Identify the concurrency problems that may occur in this code.

Duration: 25 min

Exercise 3: Starvation

In this example we showed that livelock can be produced when threads interact. In this example, they may spend all their time trying to let the other thread go first rather than doing work.

Another problem that may arise is starvation, where threads are unable to access a resource as it is always held by another thread. Think of other methods that might be used in our bank account example that could lock all accounts.

Duration: 5 min