Atomicity and Deadlock

November 29, 2007

Check-then-Act Race

```java
public void transfer (Account fromAccount,
         Account toAccount, int amount) {
    if (fromAccount.getBalance() >= amount) {
        fromAccount.withdraw (amount);
        toAccount.deposit(amount);
    }
}
```
### Good Interleaving

<table>
<thead>
<tr>
<th>Account1</th>
<th>Account2</th>
<th>Transfer 1-&gt;2</th>
<th>Transfer 1-&gt;2</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance = 100</td>
<td>balance = 1900</td>
<td>balance good</td>
<td>balance good</td>
</tr>
<tr>
<td>lock</td>
<td>balance = 0</td>
<td>unlock</td>
<td>balance bad</td>
</tr>
</tbody>
</table>

### Bad Interleaving

<table>
<thead>
<tr>
<th>Account1</th>
<th>Account2</th>
<th>Transfer 1-&gt;2</th>
<th>Transfer 1-&gt;2</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance = 100</td>
<td>balance = 1900</td>
<td>balance good</td>
<td>balance good</td>
</tr>
<tr>
<td>lock</td>
<td>balance = 0</td>
<td>unlock</td>
<td>balance bad</td>
</tr>
<tr>
<td>lock</td>
<td>balance = 2000</td>
<td>unlock</td>
<td>balance bad</td>
</tr>
<tr>
<td>lock</td>
<td>balance = 0</td>
<td>unlock</td>
<td>balance bad</td>
</tr>
<tr>
<td>lock</td>
<td>balance = 2100</td>
<td>unlock</td>
<td>balance bad</td>
</tr>
</tbody>
</table>
Atomic Check-and-Act

```java
public void transfer (Account fromAccount,
                     Account toAccount, int
                     amount) {
    synchronized (fromAccount) {
        if (fromAccount.getBalance() >= amount) {
            fromAccount.withdraw (amount);
            toAccount.deposit(amount);
        }
    }
}
```

Interleaving

<table>
<thead>
<tr>
<th>Account1</th>
<th>Account2</th>
<th>Transfer 1-&gt;2</th>
<th>Transfer 1-&gt;2</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance = 100</td>
<td>balance = 1900</td>
<td>lock</td>
<td>balance good</td>
</tr>
<tr>
<td>wait for lock</td>
<td>balance = 0</td>
<td>lock</td>
<td>balance = 2000</td>
</tr>
<tr>
<td>unlock</td>
<td></td>
<td>unlock</td>
<td></td>
</tr>
<tr>
<td>unlock</td>
<td></td>
<td>balance bad</td>
<td></td>
</tr>
</tbody>
</table>

Thursday, November 29, 2007
Deadlock

Deadlock occurs when threads are all waiting for locks and no thread can proceed.

```java
// Create first thread to repeatedly transfer in one
direction
Transferrer t1 = new Transferrer(account1, account2);
Thread t1Thread = new Thread(t1);

// Create second thread to repeatedly transfer in
the other direction.
Transferrer t2 = new Transferrer(account2, account1);
Thread t2Thread = new Thread(t2);

t1Thread.start();
t2Thread.start();
```

Interleaving with Deadlock

<table>
<thead>
<tr>
<th>Account1</th>
<th>Account2</th>
<th>Transfer 1-&gt;2</th>
<th>Transfer 2-&gt;1</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance = 100</td>
<td>balance = 1900</td>
<td>lock</td>
<td>balance good</td>
</tr>
<tr>
<td>balance = 1800</td>
<td>balance = 1800</td>
<td>lock</td>
<td>balance good</td>
</tr>
<tr>
<td>balance = 0</td>
<td>balance = 0</td>
<td>wait for lock</td>
<td>wait for lock</td>
</tr>
</tbody>
</table>
What causes Deadlock?

- Threads are holding locks while waiting for other locks
- There is a circular dependency between the holders and the waiters

Avoiding Deadlock 1

- Threads are holding locks while waiting for other locks
- Solution: don’t hold 2 locks at once.

```java
public void transfer (Account fromAccount,
                     Account toAccount, int amount) {
    boolean withdrawWorked = false;
    synchronized (fromAccount) {
        if (fromAccount.getBalance() >= amount) {
            fromAccount.withdraw (amount);
            withdrawWorked = true;
        }
    }
    if (withdrawWorked) {
        toAccount.deposit (amount);
    }
}
```
Avoiding Deadlock 2

- There is a circular dependency between the holders and the waiters.
- Solution: Make sure threads always acquire locks in the same order.

```java
public void transfer (Account fromAccount, Account toAccount, int amount) {
    Account accountToLock;
    if (fromAccount.getAccountNumber() < toAccount.getAccountNumber()) {
        accountToLock = fromAccount;
    } else {
        accountToLock = toAccount;
    }
    synchronized(accountToLock) {
        if (fromAccount.getBalance() >= amount) {
            fromAccount.withdraw(amount);
            toAccount.deposit(amount);
        }
    }
}
```

<table>
<thead>
<tr>
<th>Account1</th>
<th>Account2</th>
<th>Transfer 1→2</th>
<th>Transfer 2→1</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance = 100</td>
<td>balance = 1900</td>
<td>lock</td>
<td>balance good</td>
</tr>
<tr>
<td>lock</td>
<td>balance = 1900</td>
<td>balance good</td>
<td>balance good</td>
</tr>
<tr>
<td>unlock</td>
<td>balance = 1800</td>
<td>unlock</td>
<td>unlock</td>
</tr>
<tr>
<td>unlock</td>
<td>balance = 1900</td>
<td>unlock</td>
<td>unlock</td>
</tr>
<tr>
<td>unlock</td>
<td>balance = 1800</td>
<td>unlock</td>
<td>unlock</td>
</tr>
<tr>
<td>unlock</td>
<td>balance = 1900</td>
<td>unlock</td>
<td>unlock</td>
</tr>
<tr>
<td>unlock</td>
<td>balance = 1800</td>
<td>unlock</td>
<td>unlock</td>
</tr>
<tr>
<td>unlock</td>
<td>balance = 1900</td>
<td>unlock</td>
<td>unlock</td>
</tr>
</tbody>
</table>

Thursday, November 29, 2007