Deadlock Example with Lock Ordering

(Java Program)

COMSATS University Islamabad
Sahiwal Campus

Usama Sarwar

FA17-BS(CS)-090-B

Mr. Umer

Operating System Concepts

October 31, 2019

Table of Contents

List of Figures

Deadlock Example with Lock Ordering

I. Abstract

Deadlock describes a situation where two or more threads are blocked forever, waiting for each other. Deadlocks can occur in Java when the synchronized keyword causes the executing thread to block while waiting to get the lock, associated with the specified object. Since the thread might already hold locks associated with other objects, two threads could each be waiting for the other to release a lock. In such case, they will end up waiting forever.

A. Scenario

Transactions 1 and 2 execute concurrently. Transaction 1 transfers $25 from account A to account B, and Transaction 2 transfers $50 from account B to account A

B. Java Code

```
package deadlock;

class user {

    int AccountNumber;
    double CurrentAmount = 1000;

    user(int AccountNumber) {
        this.AccountNumber = AccountNumber;
    }
}
```

DEADLOCK EXAMPLE WITH LOCK ORDERING

```
/*    Transaction  1 transfers $25 from account A to account B */
class transaction1 {
    user A = new user(1);
    user B = new user(2);

    void withdraw(double CurrentAmount) {
        A.CurrentAmount = A.CurrentAmount - CurrentAmount;    }
    void deposite(double CurrentAmount) {
        B.CurrentAmount = B.CurrentAmount + CurrentAmount;    }

    void transaction(double CurrentAmount) {
        withdraw(CurrentAmount);
        deposite(CurrentAmount);
     }
}
/*    Transaction 2 transfers $50 from account B to account A */
class transaction2 {
    user A = new user(1);
    user B = new user(2);

    void withdraw(double CurrentAmount) {
        B.CurrentAmount = B.CurrentAmount - CurrentAmount;    }
    void deposite(double CurrentAmount) {
        A.CurrentAmount = A.CurrentAmount + CurrentAmount;    }
    void transaction(double CurrentAmount) {
        withdraw(CurrentAmount);
        deposite(CurrentAmount);
    }
}
```

DEADLOCK EXAMPLE WITH LOCK ORDERING

```
class LockTransaction {
    transaction1 t1 = new transaction1();
    transaction2 t2 = new transaction2();

    /*
    Synchronized method is used to lock an object for any shared resource.
    When a thread invokes a synchronized method,
    it automatically acquires the lock for that object and releases it when the thread completes its task.
     */

    public void Transaction1() {
        while (true) {
            synchronized (t1) {
                synchronized (t2) {
                t1.transaction(25);
                System.out.println("Transaction 1 Processing…");
                }            }        }    }
    public void Transaction2() {
        while (true) {
            synchronized (t2) {
                synchronized (t1) {
                    t2.transaction(50);
                    System.out.println("Transaction 2 Processing…");
                }
            }
        }
    }
}
```

DEADLOCK EXAMPLE WITH LOCK ORDERING

```
class DoTransaction {
    void doTransaction() {
        LockTransaction lockTransaction = new LockTransaction();

        new Thread(() -> {
            lockTransaction.Transaction1();
        }).start();

        new Thread(() -> {
                lockTransaction.Transaction2();
        }).start();
    }}

public class Deadlock {
    public static void main(String[] args) {
        DoTransaction t = new DoTransaction();
        try {
            t.doTransaction();
        } catch (Exception e) {
            System.out.println("Error: "+e);
        }
    }
}
```

C. Console Output

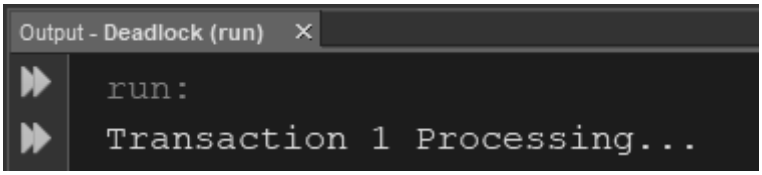

Figure 1. Deadlock Console Sample