**Tran Nguyen**

**CS 3343 Operating Systems**

**Assignment 7 12 points**

**Chapter 7**

**Due April 15**

**Email your homework to me at** [**harringp@nsuok.edu**](mailto:harringp@nsuok.edu)

**Part 1 : Use the slides or textbook definitions to answer the following (6 points)**

1. Describe the deadlock problem including the four conditions needed for deadlock to occur.

A deadlock occurs when the waiting process is still holding another resource that needs first before it can finish or when many processes try to access to the same resource same time.

Condition:

Mutually exclusive: only one process at a time can use resources

Hold and wait: a process holds at least one pending resource for additional resources held by other processes

No priority: a resource can only be released voluntarily by the process holding it, after that process has completed its task.

Loop waiting: there exists a set of {P0, P1,…, Pn-1} processes waiting so that P0 is waiting for a resource held by P1 and P1 is waiting for a resource held by P2,…, Pn - 1 is waiting for the resource held by Pn and P0 is waiting for the resource held by P0.

1. How does a resource allocation graph illustrate deadlocks?

If a process is requesting a resource, an arrow is drawn from the process node to the resource node. If a cycle is being formed, then system is in a deadlock state, and if no cycle is being formed, then system is not in a deadlock state.

1. What are the ways to prevent deadlock?

Mutual Exclusion: not required for sharable resources; must hold for non-sharable resources

Hold and Wait: must guarantee that whenever a process requests a resource, it does not hold any other resources.

No Preemption: preempted resources are added to the list of resources for which the process is waiting, or process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

Circular Wait: impose a total ordering of all resource types and require that each process requests resources in an increasing order of enumeration.

1. What are the ways to avoid deadlock?

Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.

The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.

Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

1. What are the three characteristics of a system in a safe state?

If a system is in safe state - no deadlocks.

If a system is in unsafe state - possibility of deadlock.  
Avoidance - ensure that a system will never enter an unsafe state.

1. What is the Banker’s algorithm?

Banker’s Algorithm is a deadlock avoidance algorithm. It is also used for deadlock detection. There are multiple instances, each process must claim maximum use in advance. This algorithm tells that if any system can go into a deadlock or not by analyzing the currently allocated resources and the resources required by it in the future. When a process requests a resource, it may have to wait, or when a process gets all its resources it must return them in a finite amount of time.

**Part 2: Java programming (6 points):**

Save the program as DeadlockExample.java

Run the program below. Note whether deadlock occurs. Then modify the program to add two more threads: add a class C and a class D, and call them from main. Does deadlock occur?

The code is running with a couple of warning, I can tell that Deadlock is occur in a few run.

package com.company;  
  
import java.util.concurrent.locks.\*;  
  
class A implements Runnable  
{  
 private Lock first, second;  
  
 public A(Lock first, Lock second) {  
 this.first = first;  
 this.second = second;  
 }  
  
 public void run() {  
 try {  
 first.lock();  
 System.*out*.println("Thread A got first lock.");  
 // do something  
  
 try {  
 Thread.*sleep*( ((int)(3\*Math.*random*()))\*1000);  
 }  
 catch (InterruptedException e) {}  
  
  
 second.lock();  
 System.*out*.println("Thread A got second lock.");  
 // do something  
  
 }  
 finally {  
 first.unlock();  
 second.unlock();  
 }  
 }  
}  
  
class B implements Runnable  
{  
 private Lock first, second;  
  
  
 public B(Lock first, Lock second) {  
 this.first = first;  
 this.second = second;  
 }  
  
 public void run() {  
 try {  
 second.lock();  
 System.*out*.println("Thread B got second lock.");  
 // do something  
  
 try {  
 Thread.*sleep*( ((int)(3\*Math.*random*()))\*1000);  
 }  
 catch (InterruptedException e) {}  
  
 first.lock();  
 System.*out*.println("Thread B got first lock.");  
 // do something  
  
 }  
 finally {  
 second.unlock();  
 first.unlock();  
 }  
 }  
}  
  
class C implements Runnable  
{  
 private Lock first, second;  
  
 public C(Lock first, Lock second) {  
 this.first = first;  
 this.second = second;  
 }  
  
 public void run() {  
 try {  
 first.lock();  
 System.*out*.println("Thread C got first lock.");  
 // do something  
  
 try {  
 Thread.*sleep*( ((int)(3\*Math.*random*()))\*1000);  
 }  
 catch (InterruptedException e) {}  
  
  
 second.lock();  
 System.*out*.println("Thread C got second lock.");  
 // do something  
  
 }  
 finally {  
 first.unlock();  
 second.unlock();  
 }  
 }  
}  
  
class D implements Runnable  
{  
 private Lock first, second;  
  
 public D(Lock first, Lock second) {  
 this.first = first;  
 this.second = second;  
 }  
  
 public void run() {  
 try {  
 first.lock();  
 System.*out*.println("Thread D got first lock.");  
 // do something  
  
 try {  
 Thread.*sleep*( ((int)(3\*Math.*random*()))\*1000);  
 }  
 catch (InterruptedException e) {}  
  
  
 second.lock();  
 System.*out*.println("Thread D got second lock.");  
 // do something  
  
 }  
 finally {  
 first.unlock();  
 second.unlock();  
 }  
 }  
}  
  
public class DeadlockExample  
{  
 public static void main(String args[]) {  
 Lock lockX = new ReentrantLock();  
 Lock lockY = new ReentrantLock();  
  
 Thread threadA = new Thread(new A(lockX,lockY));  
 Thread threadB = new Thread(new B(lockX,lockY));  
 Thread threadC = new Thread(new B(lockX,lockY));  
 Thread threadD = new Thread(new B(lockX,lockY));  
  
 threadA.start();  
 threadB.start();  
 threadC.start();  
 threadD.start();  
 System.*out*.println("Deadlock!!!!");  
 }  
}