CS 311 Lab – Synchronization

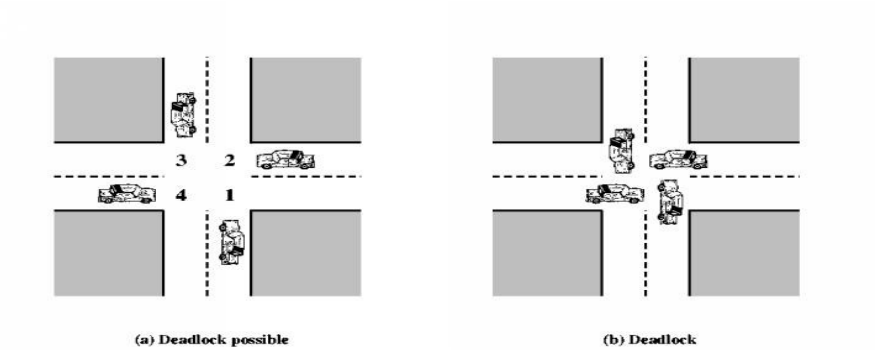
Purpose

The purposes of this lab are to 1) see a deadlock happen, 2) to employ a strategy to fix a deadlock, 3) see semaphores in action.

Deadlock

From Wikipedia: "In an operating system, a deadlock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process. If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock."

The Dining Philosophers sample code provided with this lab has a deadlock problem. In this lab, you will get to see this deadlock occur, and try to fix it.

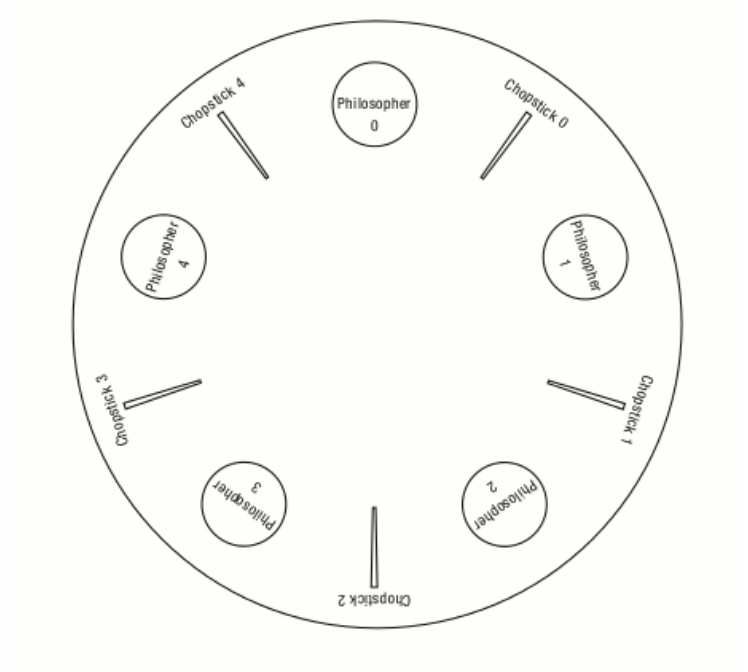


For the Dining Philosopher's Problem sample code:

1. Download, compile, and run the DiningPhilosophers.java program. Does the program deadlock? Why does it deadlock? See if you can find the evidence in the output of why the deadlock occurred. How long did it take?

2. Use the attached Dining Philosophers picture to map out which philosopher owns which chopsticks. Do you see how we have a case of "circular waiting?"

3. The provided sample code makes all philosophers pick up their left chopstick, then their right. Since they pick up the left chopstick first, we will call them "left handed". Make a copy of DiningPhilosophers.java. Call the new file DiningPhilosophersFixed.java, and try one of the strategies discussed in class to prevent deadlock.
   Hint: you can eliminate the "circular waiting" condition by making one of the philosophers be "right handed."



Semaphores

Semaphores are often used to restrict the number of threads than can access some (physical or logical) resource. Dutch computer scientist Edsger Dijkstra invented the semaphore in the early 1960s, and they have found widespread use in a variety of operating systems.

In this lab, you will get to see a semaphore in action.

First, CAREFULLY study the provided sample code and make sure you understand how it works. Ask questions if you don't understand something.

Then try the following:

The Java API docs:

1. Open the Java API docs in your browser and read about semaphores.
   https://docs.oracle.com/javase/8/docs/api/index.html
   You'll find the Semaphore class in java.util.concurrent.

The Producer Consumer Problem sample code for this lab uses semaphores:

1. Download and compile ProducerConsumerExample.java, Reader.java, and Writer.java.
   javac *.java
2. Run the code
   java ProducerConsumerExample
3. Examine the ProducerConsumerExample.java file. Find where it creates the Semaphore. What is the name of the semaphore? Find where the Consumer and Producer threads are created and started. What gets passed to the threads when they are created?

4. Examine the code and see what it is doing with the semaphore. Semaphores are used differently than locks. Notice how the Reader calls semaphore.acquire() and the Writer calls semaphore.release(). This is allowed, because unlike locks, semaphores have no notion of ownership. Which thread “acquires” the semaphore? Which thread “releases” the semaphore?

5. Add a print statement to print out the value of the number of available permits on the semaphore. (Hint: use the java api doc to help with this.) Recompile the code and watch the number of “Permits” that the semaphore has. When, where, and why does the Consumer thread wait? Comment out the “Thread.sleep()” in the Reader.java file to help see this.