Project 3 - All about Deadlock

CECS 326 - Operating Systems

Due Date:	April 4, 2023
Contributors:	@029136612 Twan Tran
	@025295541 Bharath Varma Kakarlapudi
Instructor:	Hailu Xu

1 Lab Summary

This lab involves designing an algorithm that solves the deadlock issue arising when two villages, East Village and West Village, use a single-lane road for exchanging or sharing their produce. We need to implement this solution using synchronization tools, including Semaphores and/or Mutex Locks.

1.1 Objective

The goal of this lab is to provide a solution to the mentioned problem and to demonstrate how to implement it using Semaphores and Mutex Locks.

1.2 Design

To solve the problem, we can use two locks, one for each village, and a semaphore for the road. We can represent the people from each village as separate threads: "East-village.java" and "West-village.java". Whenever a person from either village needs to cross the road, they first acquire their respective village lock. Then, the person checks the current priority (0 for East-village, 1 for West-village) to decide whether they can cross the road or need to wait. If they can cross, they acquire the semaphore for the road and release their village lock. After completing the exchange, they release the semaphore for the road and their village lock.

2 Implementation

2.1 East-village.java (Twan)

To come up with this design, we had to identify the critical section of the code, which is the part where the thread tries to acquire the Semaphore object that represents the road. To avoid deadlock, we used a Lock object for the East village to ensure that only one person from the East village can be on the road at a time. We also used a Semaphore object to represent the road and ensure that only one person can be on the road at a time, regardless of which village they are from. Finally, we used a synchronized block to ensure that the current priority of the road is checked and updated atomically, which prevents race conditions between threads.

2.2 West-village.java (Bharath)

To come up with this design, we had to identify the critical section of the code, which is the part where the thread tries to acquire the Semaphore object that represents the road. To avoid deadlock, we used a Lock object for the West village to ensure that only one person from the West village can be on the road at a time. We also used a Semaphore object to represent the road and ensure that only one person can be on the road at a time, regardless of which village they are from. Finally, we used a synchronized block to ensure that the current priority of the road is checked and updated atomically, which prevents race conditions between threads.

2.3 RoadController.java (Twan)

The RoadController class uses semaphores and mutex locks to ensure that only one village at a time can use the road and to prevent deadlocks from occurring. The idea behind this design is to have a central controller that manages the access to the shared resource (the road) and ensures that there are no deadlocks or race conditions. Using semaphores and mutex locks ensures that only one thread can access the road at any given time and prevents deadlocks by ensuring that threads do not block each other indefinitely.

2.4 Main.java (Bharath)

This Main Class is used to test this solution by creating multiple threads for each village and running them simultaneously. The program should not deadlock, and all threads should complete their execution without any issue.

3 Result

3.1 Output

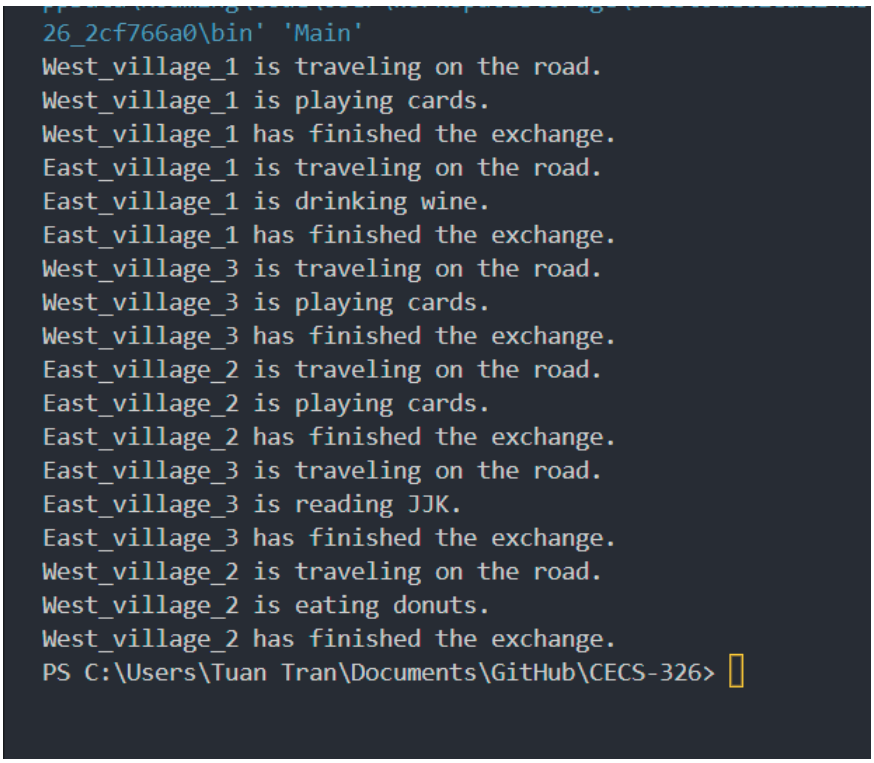

Figure 1: This is the output of the program after it ran.

We have tested the solution by creating multiple threads for each village and running them simultaneously. The program runs without any deadlock, and all threads complete their execution successfully.

3.2 Demonstration (Twan)

Link: https://youtu.be/At5G2XR7Li8