1. Look at the implementation of Semaphore::P() and Semaphore::V() in slide 6, and explain what happens in this attempt at mutual exclusion.

Semaphore s(0); // initialized to 0

// ---- thread 1 ---- // ---- thread 2 ----

T1S1: s.P(); T2S1: s.P();

T1S2: // critical section T2S2: // critical section

T1S3: s.V(); T2S3: s.V();

This code will deadlock.

s = 0;

T1S1 will attempt to decrement, but it can’t because s = 0, so thread 1 is stuck.

T1S2 will also attempt to decrement, but it can’t because s = 0, so thread 2 is also stuck.

This situation will happen regardless of whether thread 1 or thread 2 goes first.

Since both threads are stuck, the code is deadlocked.

2. Explain what happens in this version.

Semaphore s(1); // initialized to 1

// ---- thread 1 ---- // ---- thread 2 ----

T1S1: s.P(); T2S1: s.P();

T1S2: // critical section T2S2: // critical section

T1S3: s.V(); T2S3: s.V();

This code will work properly through mutual exclusion.

s = 1;

T1S1 executes P() and decrements s, then continues.

s = 0;

If T2S1 runs, then it will be blocked until T1 finishes the critical section.

T1S2 runs the critical section.

T1S3 executes V() and increments s, then stops. T2 wakes up.

s = 1;

T2S1 executes P() and decrements s, then continues.

s = 0;

T2S2 runs the critical section.

T2S3 executes V() and increments s, then stops.

s = 1;

If T2 runs first, then order of those instructions will flip, with T2 going through the cs while T1 waits. Once T2 executes V(), T1 will wake up and run through the critical section, then execute V() and stop.

3. Explain what happens in this version.

Semaphore s(2); // initialized to 2

// ---- thread 1 ---- // ---- thread 2 ----

T1S1: s.P(); T2S1: s.P();

T1S2: // critical section T2S2: // critical section

T1S3: s.V(); T2S3: s.V();

In this case, both T1 and T2 can run through the critical section concurrently.

s = 2;

T1S1 executes P() and decrements s, then continues.

s = 1;

T2S1 executes P() and decrements s, then continues.

s = 0;

T1S2 and T2S2 can both run through the critical section simultaneously.

T1S3 executes V() and increments s, then stops.

s = 1;

T2S3 executes V() and increments s, then stops.

s = 2;

Since both threads can run simultaneously, the order of threads is mostly insignificant.