CS377 Assignment 4 Write-Up

Zhanwen Chen
Vassar College

November 7, 2016

1 Version 1: Synchronized Code Blocks

1.a Implementation

I first studied the TableMon solution and implemented the Chopstick class in the fashion of fork. That is, Chopstick has an int availability variable and the code blocks would set that to 1 or 0. Then I realized it is in fact redundant because the synchronized keyword already implements a lock on the Chopstick object itself, therefore rendering the availability binary redundant. I therefore removed lines modifing chopstick[i].availability in the Philosopher class and made the Chopstick inner class completely empty.

1.b Deadlocks

When I first ran it, it deadlocked right away, but when I moved the "BURP," "thinking," and delay out of the synchronized block, nothing blocks. Meanwhile, the code is still liable for deadlocks, because monitors are like locks, and I did not implement any of the 3 solutions in Assignment 3, be it different picking-up order or a waiter.

To find out if it is opportunistic, I first ran 20 instances of the program at the same time for 10 minutes. None deadlocked.

Suspecting that it was the three lines, I moved the three lines back inside the synchronized block. It deadlocked right away. To find out if the culprit was the delay statement at the very end, I commented it out. The program again deadlocked right away. Commenting all three lines produced the same result.

I then moved just the delay statement outside the synchronized code block, and it worked again! To be sure, I commented out all three lines out and the program deadlocked right away. I concluded that having a delay statement outside the synchronized block is cheating, as it is not a dependable solution but one that that will make it work most of the time.

After realizing this, I implemented the one-phil-picks-up-differently solution as in Assignment 3 while leaving the delay statement commented out.

It worked! Although Philosophers 5 and 4 dominated the execution, 1, 2, and 3 are still visible.

2 Version 2: Synchronized Methods

2.a Implementation

I implemented the sitDown() and standUp() methods in WaiterMon, much like getForks and putDownForks() are implemented in TableMon.

2.b Thread Safety

We now have a waiter that keeps count, whose max count is 4. In addition, getLeftFork() and getRightFork() are synchronized, so the entire program is thread-safe.

After running the program in 20 terminal windows for 10 minutes, no deadlock occurred and print statements are evenly distributed across philosophers. Success!