Lab 7: The Dining Philosophers' Problem

SEG2106 A – Software Construction

Winter 2021

School of Electrical Engineering and Computer Science

University of Ottawa

Course Coordinator: Dr. Shiva Nejati

Gianluca Coletti

300065278

Pavly Saleh

300012936

Report Due Date: April 5th, 2021

Question 2.2:

Yes, deadlock would be avoided in this case since only 4 philosophers would be eating at any given time. Therefore, there will always be 2 chopsticks (at max) available for adjacent philosophers to take. The program implementing this behaviour ran without any deadlock.

Question 3.2:

Yes, deadlock would be avoided in this case since no philosopher will be waiting for a chopstick forever. A chopstick is released if the philosopher cannot grab both his left and right within a specified amount of time. This program ran with a time of 500, 1000, and 2000 milliseconds, running successfully on all specified waiting times. Once the limited waiting time passes, the philosopher either gives up on eating if they did not get the chopstick, or if they are holding a chopstick already but did not receive the second one, they release it. The program implementing this behaviour ran without any deadlock.

Question 4.2:

Yes, deadlock would be avoided in this case, since only odd philosophers start left-hand first, and even philosophers start right-hand first. There are some problems with this implementation however. The first is that this solution can cause starvation. This depends on how the thread system is implemented. This was not an issue with our implementation as the times were randomized, however, it could be on an implementation with different parameters. The second issue noticed is that the philosophers are not equal here, meaning that it seems like philosopher 4 tends to eat more than the others, as he has the advantage of always getting his first chopstick. Hence, he has an advantage over the others. Nonetheless, this is not major as the point of this exercise was not to make all philosophers equal, this was merely an observation made while the program ran. The program implementing this behaviour ran without any deadlock.