CS307 HW 2 Explanation

Samuel Lee

24774

23/11/2020

For this homework, we were asked to implement the Dining Philosophers problem in java. Firstly, I implemented the Barrier by creating a Barrier class, with a single CyclicBarrier variable. Cyclicbarrier is part of the java.util.concurrent package, and it is a synchronizer that allows a set of threads to wait for each other to reach a common execution point (aka barrier). The class was initialized to accept 5 ‘parties’ or threads, and this class was used because the homework stated that each philosopher takes different time to walk to the table, and put his plate. A random generator was used to determine how long the philosopher will take, and the cyclicBarrier.await() function was called to make sure that all 5 threads reached the barrier.

After this point, we can starting dining. I implemented separate class to handle all the operations of think, take_fork, put_fork functions inside the “statePhil” class, to also be aligned with the GUI (by initializing the Table class inside the “statePhil” class). For the think() function, it is specifically designed to not think when he is in a hungry state, and only execute in the first iteration/call of think() function (by initializing all index of state array to 0), or after he has finished eating(after eating, state[i] will be set to 0).

To prevent deadlocks and starvation, semaphores and mutexes was used. A single mutex was used for the entering and exiting of critical regions, and an array of semaphores was used to keep track of the philosophers. The general flow of the program is to first think(), then take_fork(), if successful, it will go to test() to eat. If not successful, it will remain hungry(not go back to thinking), and try to take_fork() again. If the test() is passed, philosopher(i) can eat, and will stop eating and put_forks afterwards, and go back to thinking. This is repeated infinitely.