Khadeja Iqbal 26242

CS307 Homework 2: Dining Philosophers Problem

A thread first sleeps for a random amount of time between 1-10 seconds indicating the walking time of a philosopher to the table. The thread on completing the sleep time places a plate on the table indicating the philosopher has reached the table. After this point the philosophers cannot proceed with eating and should wait for others to arrive at the table, so barriers are implemented to make sure all threads wait before the eating starts. For example, the current thread that is running is has put the plate on the table is of Philosopher 2(P2).P2 will call an Up operation on the semaphores of all other thread including threads of other Philosopher (PN where N represents 0<= N <=4).So an Up is called on the thread P0,P1,P3 and P4 which increment the value of the semaphores of these threads. Then a down operation is called on the thread but since its value is already zero it cannot be decremented further so thread P2 sleeps and a context switch happens where one of the other threads start operating in the same manner and P2 is asleep, once the value of the semaphore of P2 is incremented by one of the other threads it is ready to run again. Hence a barrier is implemented. After that the dinning phase begins, this process repeats forever. For a single philosopher there are four stages first the philosopher thinks, takes a fork, eats and keeps the fork back depending on the state (3 states: Thinking Eating and hungry) of the philosophers next to them. For understanding lets assume the currently running philosopher is P2. P2 first thinks which means that the thread P2 sleeps for a random time ranging from 1-10 seconds. P2 after thinking becomes hungry and tries to acquire forks. For implementation of taking a fork, we use a mutex and P2 acquires this mutex by making a down operation to achieve mutual exclusion. We then change the state of P2 to HUNGRY noting this fact. Now, P2 tries to acquire two forks. For this reason, another function test is implemented and the id of the process trying to acquire forks, in this case P2. The test function checks if the state of P2 is HUNGRY and then checks if the philosophers to the left and right are not in EATING state because that would mean the forks are unavailable. So, in this case for P2, test checks if its state id HUNGRY and then checks P3 AND P1 state. If P1 and P3 are not EATING that means the forks to the left and right are available for P2 to pick and move to the EATING state hence the state of P2 is changed to EATING and we make an UP call on the semaphore of P2 only if the forks are acquired. Then P2 leaves the critical region and mutex is released by making an Up call on the mutex. If the forks were not acquired the value of semaphore for P2 would be zero meaning the down operation will not be executed and the process or philosopher will be blocked, other wise we will be allowed to eat. Then we apply the GUI functions of taking fork and eating on P2. This marks the end of eating stage. Now, a philosopher P2 must decide on whether it should down its fork or not. To enter the critical region the mutex is acquired or a down call is made and the state of P2 is changed to THINKING. We test the left and right i.e P1 and P3 and only put the forks if P1 and P3 are in HUNGRY state and their respective left and rights are not EATING then we make an up call on the semaphores of P1 and P3.If the test conditions are not satisfied for either P1 or P3 then forks are not dropped by P2.Hence synchronization is achieved.