CS307 HW2 BARRIER AND DINING ALGORITHM Cem Ertürkan 26801

Dijkstra's philosopher analogy is a description of a situation where there are 5 philosophers sitting around a circular table and in front of them there are their plates of spaghetti one for each and philosophers have 3 different possible states, hungry, eating and thinking. The challenge of this problem is that there are only 5 forks and to eat spaghetti a philosopher needs 2 forks which means that there needs to be a mechanism/order(random in our case) in which philosophers transition between three states picking up forks and putting them down.
This project is an implementation of Dijkstra's 5 philosophers problem with graphic output.

In this project each philosopher is represented with a thread run concurrently under OS supervision. Therefore, the words "thread" and "philosopher" will be used interchangeably in this document.

In addition to the original problem, the project also simulates the philosophers walking to the table to sit with randomly generated walking times(0-10 secs) (lines 327-328) through the making the dedicated thread go to sleep for the randomly generated time duration. After that walking time each philosopher puts their plates on the table(yellow plates) and starts to wait. There is a 5 thread barrier implementation to simulate philosophers waiting for others to also arrive at the table(none can go through unless all sit). When all arrive, they start dining.(plates turning into black and white)

After the barrier, a while loop starts and keeps on going as long as the program is run. At the beginning of the loop, all threads are assigned a random sleeping time(0-10secs)which the philosophers spend to think. Following the thread's sleep, a down operation on the general mutex is performed to keep any race condition from happening while accessing and altering the shared resources(forks in our analogy). After that, the philosopher's state changes to hungry. The following if statement makes sure that none of the philosophers adjacent to one that is executing the statement are eating(which also means they don't have any forks in their hands) and the executing thread is hungry(ready to eat) before making a philosopher take two forks besides their plates and assigning them to be in eating state. Within this if's scope there is also a up statement on the dedicated semaphore which will later be downed. Mutex is upped and the shared resources become open. If that if statement is not satisfied the statement since there is no up on the thread dedicated semaphore, the following down statement will block that thread until one of the other threads ups that mutex. Say the statement was satisfied then that philosopher's plate turns blue indicating that it is eating. Since the thread will again access shared resources by putting the forks down and changing to thinking state there is again a down on mutex. Then the thread checks if its left thread is hungry and if itself and the left of its left thread are not eating, if so it updates the thread's state as eating and ups the left thread's semaphore which then makes the left thread able to continue executing breaking out of its blocked state. The same thing is also done to the right thread. At the very end, the mutex is released and the thread goes back to thinking visually(black and white plate).

This loop keeps on going for all concurrently run threads.