Q1 Documentation

- 4 program files are attached:

For a1,a2,b1,b2 parts respectively.-

- Working of the code:

We are making an array of forks which represents all the forks kept from indices 0 to 4. Each element in array of forks is initialized to 0, which represents that the fork is available at this point

Then in a function(called by main) we create a philosopher thread for each philosopher kept from indexes 0 to 4. Then we use pthread_create() and pthread_join() so that they finally run the finish function, which stimulates the process further.

Here, The philosopher function has an infinite loop that simulates the philosopher thinking and then attempting to eat.

To eat, the philosopher must pick up the fork to their left and right. The order in which the forkss are picked up depends on the philosopher's number.

Philosophers with even numbers always pick up the right fork first, while philosophers with odd numbers always pick up the left fork first. This order is implemented to prevent the possibility of a deadlock occurring.

Once the philosopher has picked up both forks we set forks to be 1, they eat for a short period of time and then put the forks back down and there we set the value of fork for that index to be 0 again. The process then repeats, with the philosopher thinking and then attempting to eat again.

There are some minor changes for a1, a2, b1, b2 but this is the process majorly going on in them.

In a1 and b1 we are following strict ordering so we are making 5 functions 1 for each object in philospher's thread array and in a2 and b2 we are using semaphores for using synchronization happen. (sem_wait, sem_signal and sem_trywait (in b2),(the sem_trywait() function locks the semaphore referenced by sem only if the semaphore is currently not locked))

In b1 and b2 we have an additional array bowls with 2 elements 0 and 1 which represent 2 sauce bowls. They are assigned to a philosopher according to the need in each program.

Other functionalities remain similar for a1,b1 and a2,b2 respectively.
