BONUS

Documentation

How the program works / Logic of the program :

1. **PART 1:**
    a. For the first part, I first made an array of 5 philosophers and an array of threads representing each of the philosophers.
    b. Then the forksWait function is called which calls the sem_init function on the 5 forks that are present on the round table.
    c. After the sem_init is called on forks, I then call the philoThreadCreate function which calls the pthread_create function for each philosopher thread ID and the philosopher.
    d. The function assigned to each thread is the philosopher function which uses sem_wait() and sem_post() functions to prevent deadlock and also enable all the threads to complete the task assigned which is to execute the eat() function and also finish eating. The solution is that each philosopher from philosopher number 1 to 4 picks up the immediate left fork on the table and philosopher number 5 picks the right fork. Since the right fork of philosopher five is the left fork of philosopher 4, philosopher 5 has to wait for both forks while everyone else has at least one fork each. No deadlock will be achieved in this solution because at all times, there will be someone who is eating and when he finishes his forks will be up for grabs to the rest of the philosophers.
    e. Once all the threads have executed the philosopher function then the philoThreadJoin function is called and it joins all the 5 threads corresponding to each of the 5 philosophers sitting on the round table.

2. **PART 2:**
    There will not be a deadlock case if all philosophers need only one fork to eat and can access one of the 4 sauce bowls because there will not be a case wherein there is a race for some resource like forks or bowls. There are enough resources for all 5 threads to execute without any deadlock condition.

3. **PART 3:**
    a. In this part, I made counting semaphore bowls which have been initialized to 4 and 5 binary semaphores named forks same as in question 1.
    b. Now in this question to avoid deadlock we first make the philosopher wait for a bowl before they start picking the forks, hence at a time only 4 philosophers will have bowls and only they can pick fork, now since 5th philosopher doesn't have a bowl and he cannot pick up any fork, hence he waits, now the 5th fork left will be used by one the existing processes, now, in that case, that philosopher can start eating, hence we avoided deadlock where each process waits for another to execute.

Run Instructions: Run the make command to generate 2 executables(philo1 & philo3).