Readme

Question_1
Part A:
In this question 1 part a there are 5 philosophers sitting in the round table and in order to eat each one have to use 2 forks in front of them that is present such that a single philosopher do only 4 things in a cycle i.e,

Think->pickupforks->eat->putdownforks->Think

Input:
No input need to be given in this question_1 Part A

Output:
1) In 1st variant i.e, using strict ordering of the resource requests we need to avoid the deadlock condition i.e, just by using the non synchronisation primitives

2) In 2nd variant i.e, using utilization of semaphores to access the resources I.e, using the synchronisation primitives.

Data Structures used:
In 1st variant
pthread_t array - This array has been used to store the 5 threads representing all five philosophers that we have created and run parallely.

Int array - This array has been used to store the forks condition if 0 means that it is free to use and 1 means already occupied.

Data Structures and Synchronisation Primitives used:
In 2nd variant
pthread_t array - This array has been used to store the 5 threads representing all five philosophers that we have created and run parallely.

sem_t array - This array of 5 elements is here used to store the semaphores each representing the availability of that fork in the array on which they are indexed.

sem_t variable: A lock to lock the process of picking up the forks such that at one time only one philosopher can undergo the process of choosing the fork at one time.

Description explaining how the programme works:

First, we have created the five threads each representing the five philosophers.

Then we create a semaphore named lock as well as a semaphore array representing forks which will ensure that there can be only one philosopher on the table who can go through process of picking up forks so that deadlock can't occur as the lock will help handing over

either both the forks or nothing to the philosophers at a time so the condition of deadlock is thus automatically removed.

Then we create five semaphores each representing the 5 forks and thus creating an infinite loop such that the dining of Philosophers can go upto infinity.

Question_1
Part B:

In this question 1 part b there are 5 philosophers sitting in the round table and in order to eat each one have to use 2 forks in front of them and any 1 bowl out of the 2 bowls on the table whichever is free at that time such that a single philosopher do only 6 things in a cycle i.e,

Think—> pickup forks —> pickup bowl —> eat —> putdown forks —> putdown bowls —> Think

Input:

No input need to be given in this question_1 Part A

Output:

In here I have done this by using t utilization of semaphores to access the resources I.e, using the synchronisation primitives.

Data Structures and Synchronisation Primitives used:
In 2nd variant

pthread_t array - This array has been used to store the 5 threads representing all five philosophers that we have created and run parallely.

sem_t array - This array of 5 elements is here used to store the semaphores each representing the availability of that fork in the array on which they are indexed.

sem_t array - This array of 2 elements is here used to store the semaphores each representing the availability of bowl that are on the table.

sem_t variable: A lock to lock the process of picking up the forks such that at one time only one philosopher can undergo the process of choosing the fork and bowl at one time.

Description explaining how the programme works:

First, we have created the five threads each representing the five philosophers.

Then we create a semaphore named lock and semaphore array representing forks and semaphore array representing the bowls on the table which will ensure that there can be

only one philosopher on the table who can go through process of picking up forks as well as bowl so that deadlock can't occur as the lock will help handing over either both the forks and bowl or nothing to the philosophers at a time so the condition of deadlock is thus automatically removed.

And then wecreating an infinite loop such that the dining of Philosophers can go upto infinity.