Problem 1 Solution Description:

My program has 3 files corresponding to the 3 cases of the dining philosophers case.

Launching Threads(Common for all cases):

To ensure that there are 5 philosophers in the main process I have created an array of pthread_t and have simultaneously started all the threads using pthread_create and at the end to safely end all the threads pthread_join is called.

Case1:

Case 1 is the standard dining philosophers problem.

To achieve thread synchronization here I have used an array of binary semaphores of size 5(forks[5]) with each semaphore mapped to a fork.

So whenever a philosopher(thread) wants to eat it first picks up the left fork using function sem_wait(&fork[left]) which provides access to the left fork if it is available or puts the philosopher in thinking (waiting) state till it can successfully get the access.

Similarly, after getting the left fork the philosopher tries to get the right for using sem_wait(&fork[right]) which provides access to the right fork if it is available or puts the philosopher in thinking (waiting) state till it can successfully get the access.

After having both the forks philosopher goes into eating state and leaves the forks after some time.

This continues till all the philosophers are fed (this is done using an int array to store the number of meals and each philosopher can eat he\she has eaten 10 meals).

Note: This solution has a deadlock as whenever all 5 philosophers will pick up their left fork then no philosopher will be able to pick up their right fork and a deadlock occurs.

To solve this problem we change the way philosopher 4 picks up the forks, instead of taking the left fork first philosophers 4 takes the right fork first and left fork afterward.

This ensures that even if philosophers 0,1,2,3 takes up their left fork philosopher 4 will try take its right

fork(philosopher's 1 left fork) and will not be able to take it and philosopher 3 can take both the forks and start eating and therefore no deadlock will occur.

Case2:

Case 2 is the modified dining philosophers problem. Where each philosopher needs a single fork to eat and there are only 4 soup bowls.

To achieve thread synchronization here I have used an array of binary semaphores of size 5(forks[5]) with each semaphore mapped to a fork and a counting semaphore for allocating soup bowls(bowl).

Each philosopher first tries to take a soup bowl using sem_wait(bowl) which provides access to the bowl if it is available or puts the philosopher in thinking (waiting) state till it can successfully get the access.

After which philosopher tries to pick up the left fork using function sem_wait(&fork[left]) which provides access to the left fork if it is available or puts the philosopher in thinking (waiting) state till it can successfully get the access.

After having the bowl and the left fork philosopher goes into eating state and leaves the fork and bowl after some time.

This continues till all the philosophers are fed (this is done using an int array to store the number of meals and each philosopher can eat he\she has eaten 10 meals).

Note: There was not any special need for having the 5 binary semaphores for forks as each philosopher only needs one fork and is mapped to its left fork. But having semaphores only provides safety to our code as two philosophers will not be able to share the fork.

Clearly, no deadlock will occur as each philosopher will get a turn to eat when it has the bowl and it has the access to the forks each and every time.

Case3:

Case 3 is the modified dining philosophers problem. Where each philosopher needs both forks to eat and there are only 4 soup bowls.

To achieve thread synchronization here I have used an array of binary semaphores of size 5(forks[5]) with each semaphore mapped to a fork and a counting semaphore for allocating soup bowls(bowl).

Each philosopher first tries to take a soup bowl using sem_wait(bowl) which provides access to the bowl if it is available or puts the philosopher in thinking (waiting) state till it can successfully get the access.

After which philosopher tries to pick up the left fork using function sem_wait(&fork[left]) which provides access to the left fork if it is available or puts the philosopher in thinking (waiting) state till it can successfully get the access.

After getting the left fork the philosopher tries to get the right for using sem_wait(&fork[right]) which provides access to the right fork if it is available or puts the philosopher in thinking (waiting) state till it can successfully get the access.

After having the bowl and the left fork philosopher goes into eating state and leaves the fork and bowl after some time.

This continues till all the philosophers are fed (this is done using an int array to store the number of meals and each philosopher can eat he\she has eaten 10 meals).

Note: There will not be any deadlock in this case as the philosophers pick up the bowl first. So at any given stage, there can be a maximum of 4 philosophers with a bowl and there will be 5 forks available so at least 1 philosopher will be able to eat, and hence this will not result in a deadlock.

Nakul Thureja

2020528