ASSIGNMENT 4 WRITE_UP

NAME : SUYASHI SINGHAL
SECTION - B
ROLL NO - 2019478

→ I have basically implemented two variants of the dining philosophers problem.
→ One is using blocking semaphores while the other is using non blocking semaphores

LOGIC and TECHNIQUE
→ In my code I basically allow only one philosopher to be able to eat at any point of time by checking what the other philosophers are doing. If the other philosophers are not eating, then the philosopher can acquire the bowls and forks and eat.
→ Otherwise, the philosopher would have to wait for the other philosophers to release the resources, namely the bowls and the forks.
→ A philosopher can only eat when he has acquired both the bowls and his left and right forks. Thus, we make use of mutual exclusion as otherwise, it can lead to deadlocks.

Description and functionality of code
1) Blocking
   a) WAIT - Blocking wait basically decreases the value of the semaphore by 1 and blocks in case the resulting value is negative.
      Value is decremented only when a lock is acquired by a thread.
      Also, for the blocking, we use the pthread_cond_t condition variable as it blocks according to a conditional variable.
   b) SIGNAL - It basically increases the value of the semaphore by 1 and unblocks if other processes are waiting in the queues for the resource to be released.
      Also, at the same time the condition variable is signalled which basically unblocks the lock.
2) Non Blocking
   a) WAIT - We use the non blocking function like trylock in the pthread library to implement this wait.
      This type of wait tries to acquire the lock using trywait. If it is successful, it decreases the value of semaphore by 1. It returns 0, if the resultant value of semaphore is positive, else it returns -EINVAL .
      It returns a value instead of blocking the thread. The thread can then decide on the basis of this value to enter into the critical section or not.
   b) SIGNAL - We use the non blocking function like trylock in the pthread library to implement a non blocking signal.

This type of signal tries to acquire the lock using trywait. If it is successful, it increases the value of semaphore by 1. It returns 0, if the resultant value of semaphore is positive, else it returns -EINVAL .
It returns a value instead of signalling the conditional variable.

3) signal_debugg()

This function is used as a debugger to see the values of semaphores and their behavior and determine whether there would be a deadlock or not.

DESCRIPTION

- In both the codes, the function dinner() is the main function executed by all the threads. In this function, the thread initially tries to acquire a lock to the global mutex. It then changes the state of the philosopher to hungry.
- The philosopher enters the check() function if the resources are available. In case no other philosopher is eating, i.e bowls are available, he starts eating. We then update his state to be thinking. He then signals and relinquishes the resources held by him and signals the global mutex.
- However if the philosopher did not have the resources, he waits to get those resources using the wait() call.
- After this happens, the philosopher thread again enters the global mutex section and checks the state of his left and right philosophers which execute the check() function.
- In this code, basically we are making sure that only one philosopher can be eating at a time and no other philosopher can eat when another philosopher is eating.

THANK YOU !