Assignment 4

Tushar Mohan
2019393

Functionality :

This code is nothing but a follow up of the philosopher's problem implemented in the Quiz2. The main difference comes in the definition of semaphore. The semaphores are implemented via the pthread library in C.

The same functions have been implemented by me via the pthread library which were primarily imported from semaphore.h header file.

sem_init(sem_t *s, int value) - which is used to initialize the semaphore passed as argument

sem_wait(sem_t *s) - which is used to put a blocking lock on the resource to be used.

sem_post(sem_t *s) - which is used to release the blocking lock from the resource passed in as parameter.

sem_wait_nonblock(sem_t *s) - which is used to put a non-blocking lock on the mutex of the semaphore.

sem_post_nonblock(sem_t *s) - which is used to release a lock from the mutex of the semaphore.

philo(void * num) - is the function which actually does the job of philosophers' problem via blocking lock on semaphores. This is passed on to the threads.

philo_non_blocking(void * num) - is the function which actually does the job of philosophers' problem via non-blocking lock on semaphores. This is passed on to the threads.

There is also Deadlock avoidance in the code. This is done by alternating assigning the forks to different philosophers. If the number of the philosopher is even he is first provided with the fork on its right then on left. Whereas if the number of the philosopher is odd, then he is first provided with the left fork and then the one on right. This way there is no resource is in such a way where multiple threads are trying to put a lock but are not able to.

References :

- https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread_cond_wait.html
- https://www.qnx.com/developers/docs/6.5.0SP1.update/com.qnx.doc.neutrino_lib_ref/p/pthread_mutex_trylock.html

- https://repository.middlebury.edu/islandora/object/scholarship%3A1357/datastream/OBJ/view
-