OS-Assign4

I implemented "my_semaphore" struct which needs 3 fields: count,mutex,condition variable

- int type value to store value of the semaphore
- **pthread_mutex type variable m** to ensure locks in semaphore to ensure mutual exclusion for critical section issues
- **pthread_cond type variable cv** to ensure there is no starvation and critical section issues without wasting CPU cycles(also avoids deadlock)

Two primitives are :**wait,signal** -both are wrapped using lock to ensure only one thread is there.
init function initialises the fields with default values.
signal_print_s prints the current value of the semaphore .
For the blocking variant I have locked using *pthread_mutex_lock* and for the non-blocking variant I have used *pthread_mutex_trylock* rest all the code is the same.
My algorithm is working fine for both the variants

- signal_s(blocking) /signal_ns(non_blocking)-
  This is responsible for incrementing the count and to wake up any sleeping thread inside the condition variable.For efficiency we call *pthread_cond_signal* only when there are waiting threads
- wait_s(blocking) /wait_ns(non_blocking)-
  This is responsible for decrement of the value only when its not zero ,till then it waiting inside the condition variable and mutex is unlocked so
  that other threads can enter.

Dining philosophers i have implemented in two ways

- Philosophers are Semaphores- here i check whether my neighbor is eating or not and then check both the bowls are available or not
  and then proceed to eat.
  Once i am done eating i notify my neighbors i completed if any one was hungry they will eat.
- Resources(forks,bowls) are Semaphores- here i check whether the forks,bowls i need are there or not ,if present i will eat or else will wait in condition variable.
  Once I'm done eating I notify that resources are available.
  In both the implementation only one philosopher is allowed to enter and eat since both the bowls should be available simultaneously to eat.

SOURCES-https://docs.oracle.com/cd/E19683-01/806-6867/6jfpgdcnd/index.html
geeksforgeeks articles, http://cs241.cs.illinois.edu/coursebook/Synchronization#mutex, wikipedia,
http://www.cs.kent.edu/~ruttan/sysprog/lectures/multi-thread/multi-thread.html#thread_condvar_example
https://cs61.seas.harvard.edu/wiki/images/1/12/Lec19-Semaphores.pdf
https://github.com/LoyolaChicagoBooks/operatingsystems/blob/master/source/deadlock.rst

https://legacy.cs.indiana.edu/classes/p415-sjoh/hw/project/dining-philosophers/index.htm
https://xspdf.com/resolution/54548535.html