User Thread Library

Names and Net IDs of the members of the group:
- Shikha/ sv629
- Praveen/pp813
- Omkar/osd14

iLab M/c: ilab1

- Contains the following files:
  - main.c – Complex Mutexes, testing with our library
  - bench.c – testing with pthread library
  - bench_linear.c – Running functions without threads
  - simple_bench.c – Without Mutex and deadlock with our library
  - my_pthread.c
  - deadlock_my_pthread.c – deadlock code inclusion with our library
  - my_pthread_t.h
  - list.c
  - list.h
  - queue.c
  - queue.h

- Rationale: (Assumptions and flow)
  - Each thread could be in one of the 4 states -> READY, RUNNING, WAITING, FINISHED.
  - Each queue of MLFQ contains threads which are in READY state.
  - A separate waiting queue, finished queue contains threads in WAITING, FINISHED states respectively.
  - A thread in RUNNING state will not be in any of the queue.
  - A **dynamic** timer signal is setup with scheduler as signal handler.
  - Mutex structure contains:
    - the id of the lock acquired thread
    - id's of the threads waiting for the mutex and
    - Each thread waiting for a mutex contains the id of the lock acquired thread.
  - If a thread A either joins another thread B or waits for mutex that is acquired by another thread C, then it waits in the WAITING queue.
  - As soon as the thread B exits or thread C releases the mutex, then thread A is pushed to one of the READY queues based on its priority.

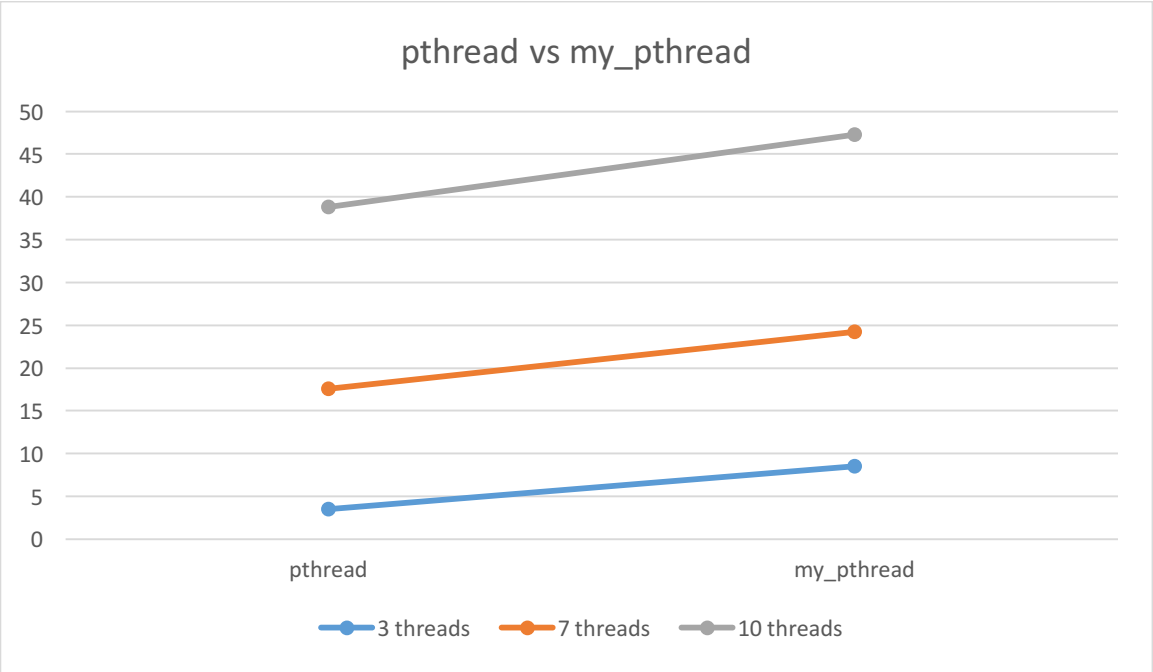
- Working of the Scheduler:
  - The scheduler has a multilevel priority queue for scheduling.
  - There are 3 levels of the priority i.e. 0, 1, 2. The priority of the listed levels are in ascending order with a time slice of 50, 100, 200 milliseconds respectively.
  - Initially each thread is added to the highest priority in the queue. If a thread finishes an assigned time quantum, the thread's priority is decreased and pushed to a lower priority queue.
  - Signal Handler: Picks up the next thread in the queue post the expiration of quanta or completion of execution.
  - Maintenance Cycle: After a period of time of every 1000msecs, based on the following checks few solutions have been implemented

- Maintenance Cycle uses:
  - Problem - Starvation / Solution – Ageing

- During every maintenance cycle, if a thread is sitting in the queue for more than 1000msecs, the thread is moved from lowest priority to highest priority.
- If they do not complete running in their quanta, they are moved to the lowest priority, the cycle goes on till completion.
- This avoids starvation

- Problem - Priority Inversion / Solution – Priority Inheritance
  - When a thread tries to lock a mutex that is already being locked by another thread, then that id is noted.
  - During the maintenance cycle, priority is checked with respect to the mutex. If that thread exists, then the thread having the mutex inherits the property of the thread waiting for the mutex.
  - When such an occurrence happens, the transfer of mutex's holding from lower to higher priority occurs, and helps them run to completion.

➢ Comparison with benchmark:
- The comparison is done with a standard pthread library and shown below as a graph.



➢ Deadlock Detection and prevention –
- We have attempted to configure deadlock prevention, it works for basic condition, i.e. circular wait(locking 2 mutexes) in case of 2 threads.
- We haven't added it to the my_pthread.c file but adding it to deadlock_my_pthread.c.

P.S.: It wasn't specifically asked, but we encountered deadlock during multiple mutexes, so tried to detect and resolve the deadlock.