Shashikanth Rangan P  : srp197
Shreyas UM : sm1717
Amith Vishwakarma : av475
ILAB machine : cd.cs.rutgers.edu

# Assignment 1

**Specifics of Pthread Library**

*my_pthread_create(my_pthread_t* pthread, void (*func)(void))*
Creates a thread specified by “pthread” and the function passed. A thread’s context is made using makecontext() which takes function pointer as an argument.
Example usage : my_pthread_create(&squart,&square_root)

*my_pthread_join(my_pthread_t thread,void **value_ptr)*
Waits on the thread specified by “thread” to complete before it begins executing. We have a parameter called wait which is set to 1 until “thread” finishes which is indicated by a parameter called “active”.
Example usage : my_pthread_join(squart,NULL)

*my_pthread_yield()*
Explicit call to the scheduler to give up a thread’s CPU time slice and to be put back into the ready queue. When the thread yields we swap the context back to the scheduler which schedules the next thread.

*my_pthread_exit(void *value_ptr)*
This causes the thread to terminate. We set the parameter “active” to 0.

*my_pthread_mutex_init(my_pthread_mutex_t *mutex)*
Initialises a mutex value.

*my_pthread_mutex_lock(my_pthread_mutex_t *mutex)*
If threads tries to lock on a mutex and mutex value is 0, set mutex value to 1 and continue execution. A next thread requesting for this mutex is put into a wait queue.

*my_pthread_mutex_unlock(my_pthread_mutex_t *mutex)*
Sets mutex value of variable specified by “mutex” to 0. Scans through waiting queue and puts a thread waiting on this mutex to the waiting queue.

*my_pthread_mutex_destroy(my_pthread_mutex_t *mutex)*
Sets mutex pointer to NULL.

Prior to implementing a multilevel feedback queue we implemented one running queue and one waiting queue. The running queue is scheduled using FCFS.

If a thread waits on a particular mutex it is pushed to the waiting queue. Once the mutex is set to 0, we take a thread waiting on that mutex and put it into the ready queue. All the queues are implemented as array of pointers.

## **RUNNING A PERIODIC INTERRUPT**

We ran a 50ms timer using setitimer() function and used a signal handler to call the scheduler to schedule the next thread. The scheduler then context switches to the next thread in line.
For specifics see : *main()*

## **MULTILEVEL PRIORITY QUEUE**

We further extended our running queue into a multilevel priority queue. To each thread we added a parameter time_allotted and priority. We set the initial priority to 7(Highest Priority) and time_allotted to 0.5ms. When each thread was running we set an interrupt based  on the value specified by time_allotted. We then passed control back to the scheduler which then reduced the priority by 1 if time exceeded time_allotted.
If the priority was lesser than 6 we pushed it to the lower queue and increased time_allotted to that thread. (time_allotted += 5 (ms))
For specifics see : *void my_pthread_yield()*

## **PERIODIC MAINTENANCE**

We periodically scanned through the threads in the lower queue and if the value of the running thread's time was above 1.5 ms we increased the priority of the thread and pushed it back to the upper thread.
For specifics see: *void time_handler()*

## **Benchmarks**

Functions used for benchmarking :
*void thread1() :* simple iteration from 1 to 10
*void fibonacci()* : calculating fibonacci numbers from 1 to 10
*void squares()* : calculating squares from 1 to 10

*void increment_count()*: incrementing count till 1000 (2 threads). (lock required).
*void square_root()*: calculating square root of numbers from 1 to 10
*void cubes()* : calculating cubes from 1 to 10
*void cube_root()*: calculating cube roots from 1 to 10
*void get_count()*: gets count value (2 threads) (requests lock)

1: Total of 10 threads,4 requesting a mutex:
   Execution time : 0.000652 secs
2: Total of 10 threads,4 requesting a mutex and all calling my_pthread_yield():
   Execution time : 0.007014 secs
3: Total of 10 threads,4 requesting a mutex and all calling sleep(1):
   Execution time : 2.908045 secs

**Stepping it up**

*void thread1() :* simple iteration from 1 to 50
*void fibonacci()* : calculating fibonacci numbers from 1 to 50
*void squares()* : calculating squares from 1 to 50
*void increment_count()*: incrementing count till 1000 (2 threads). (lock required).
*void square_root()*: calculating square root of numbers from 1 to 50
*void cubes()* : calculating cubes from 1 to 50
*void cube_root()*: calculating cube roots from 1 to 50
*void get_count()*: gets count value (2 threads) (requests lock)

1: Total of 10 threads,4 requesting a mutex:
   Execution time : 0.001062 secs
2: Total of 10 threads,4 requesting a mutex and all calling my_pthread_yield():
   Execution time : 0.010296 secs
3: Total of 10 threads,4 requesting a mutex and all calling sleep(1):
   Execution time : 14.908640 secs

## COMPILATION

```
gcc main.c my_pthread.c -lm
```