<u>Pthread library implementation and Scheduler</u>

Below is a description of the necessary structures and protocols needed to make a user thread implementation of the pthread library functions;

```
int my_pthread_create( my_pthread_t * thread, my_pthread_attr_t * attr, void *(*function)(void*),
void * arg);
void my_pthread_yield();
void my_pthread_exit(void *value_ptr);
int my_pthread_join(my_pthread_t thread, void **value_ptr);
int my_pthread_mutex_init(my_pthread_mutex_t *mutex, const my_pthread_mutexattr_t *mu-
texattr);
int my_pthread_mutex_lock(my_pthread_mutex_t *mutex);
int my_pthread_mutex_unlock(my_pthread_mutex_t *mutex);
int my_pthread_mutex_destroy(my_pthread_mutex_t *mutex);
```

Structures

<u>The Ready Queue</u>

In order to track the scheduling and priorities of running threads we made a ready queue. The ready queue is a multi level queue where each thread that is either executing or ready to be executed is inserted at some point. Each level of the queue has a priority associated with it that determines how frequently each thread at that level is run. Also, each level has a multiplier that determines how long each thread in the queue runs until the scheduler is run again. Each level in the ready queue is represented by a round-robin queue linked-list. There are a total of 10 subqueues in the ready queue. As the level of the subqueue increases, the priority of that level decreases and the multiplier associated with that level increase.

<u>The Join List</u>

This linked list contains all the threads that have called my_pthread_join() and are waiting for their desired thread to exit. When a thread calls my_pthread_exit() it searches this list to see if any matching thread that called join on it exits. If so, the join is complete. If not the calling thread enters the Exit List.

<u>The Exit List</u>

This linked list contains all the threads that have called my_pthread_exit(). The threads that call my_pthread_join() search through this queue to see if their desired thread has already exited. If so, they continue executing. If not, they are enqueued into the Join List.

The Waiting Queue

Each mutex struct contains a waiting queue. This queue is for the threads that are waiting for that mutex to be unlocked. When the mutex is unlocked, the first thread in the waiting queue is dequeued and scheduled. The mutex is again locked when the thread is scheduled.

The Scheduler

This function maintains the ready queue, and determines which thread should begin or stop its execution. This function saves the context of the current thread, and by using the properties of the ready queue as stated above, it determines which thread will be executed next and for how long. Threads that have been scheduled many times will have their priority decreased (sub-queue level increased). Each time a thread is run, if it is eligible for rescheduling (its not in the Join, Exit, or Waiting queues) it is demoted a priority level. The scheduler is run after a timer expires, which is governed by the multiplier from the priority level of the thread that was last run.

To prevent starvation, after every 100 times the scheduler is run, it checks to see if there are any threads in the last subqueue of the ready queue. If so, it moves all the threads to the top level of the ready queue. This ensures that the oldest threads will not be forgotten.

Analysis

The main property of our scheduler that we experimented with was the time multiplier given to each ready queue level. At first we incremented the multiplier by one for each increase in the subqueue level. However, we noticed that as threads reached the bottom of the ready queue, there was too much time allocated for them (half a second). This made performance pretty slow. So instead we designated 3 multiplier zones in the ready queue. For levels 1 – 3 is was 1, levels 4-7, it was 2, and 8 – 10 it was 3.