CS416 - Assignment 1

Names: Nicholas Prezioso, Benjamin Cahnbley, and Marcella Alvarez

Username of iLab: njp107, bc499, and ma1143

iLab Server: flyweight

*ALL Critical Sections within my_pthread_create cannot be interrupted by anything, including the scheduler (Noted by _CRITICAL_)

Methods:

void enqueue(my_pthread* t);

Enqueues thread into the correct queue by priority

my_pthread* dequeue();

Find a queue to dequeue, starts with searching the highest priority queue and then continues downwards.

void thread_init();

Initializes my_pthread, called the first time my_pthread_create is called, and is only done once.

int my_pthread_create(my_pthread_t * thread, pthread_attr_t * attr, void *(*function)(void*), void * arg);

Creates a thread and adds it to the highest priority queue

void scheduler();

In charge of changing threads and time slices

void my_pthread_yield();

Call to the scheduler requesting that the thread (AKA context) can be swapped out for another

void pthread_exit(void *value_ptr);

Call to end the pthread that called it. Saves value_ptr

int my_pthread_join(my_pthread_t thread, void **value_ptr);

Call making sure that the parent/calling thread will not continue until the one given is done.

Saves value_ptr

int my_pthread_mutex_init(my_pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr);

Initializes a mutex created by the calling thread, status noted as MUTEX_UNLOCKED

int my_pthread_mutex_lock(my_pthread_mutex_t *mutex);

Locks the given mutex, noted by MUTEX_LOCKED

int my_pthread_mutex_unlock(my_pthread_mutex_t *mutex);

Unlocks a given mutex, noted by MUTEX_UNLOCKED

int my_pthread_mutex_destroy(my_pthread_mutex_t *mutex);

Destroys a given mutex, noted by MUTEX_DESTROYED

void interrupt_handler(int sig);

Calls the scheduler as long as no thread is in a __CRITICAL__ section

void markDead();

Marks thread as dying, status noted as THREAD_DYING

Structure:

There are queues for each priority level-- high (queue3), medium (queue2) and low (queue1). Their context is saved, and each running thread is identified by current_thread. This is the thread struct that changes contexts to run other threads. There is an interrupt which interrupts after a certain amount of time to go to the handler, which goes to the scheduler. Every time a thread is descheduled, its priority is decreased by one. Aged threads eventually have their priority increased. Lower priority threads have larger time slices.

Testing:

We tested our program with multiple testing programs (not including the Benchmark), and they were successful. First, a basic thread where one thread is created and did a function. This is where we fixed our initial problems. We then made more complicated testers to pass different scenarios, and our program worked for all of them. For testing Benchmark, our program works for all as well.