Asst1: Adventures in Scheduling

Scheduler

The following three macros defined in `my_pthread.c` determine the scheduling behavior:

- `NUM_PRIORITY_LVLS`: An integer that defines the number of priority queue levels to create.
- `INTERRUPT_TIME`: A number that defines the interval time in microseconds in which the scheduler should run. For this assignment, the value must be 25000.
- `BASE_TIME_SLICE`: A number that defines the time slice of the threads in the highest priority queue.

The scheduler creates `NUM_PRIORITY_LVLS` queues setting the time slice of the highest priority queue to `BASE_TIME_SLICE` microseconds. Each subsequent lower priority queue is set a time slice two times the time slice of the priority level immediately above it. Every `INTERRUPT_TIME` microseconds, the scheduler runs the next highest priority thread.

All new threads are initially given highest priority. When a running thread doesn't finish execution after it has completed its current prioritized timeslice, its priority is decreased one level before being placed back in the queue. To avoid starvation, a maintenance cycle is run, increasing the priority of all threads that haven't run for a while.

When thread's exit, their return value is saved in their TCB. If a thread is waiting to join with the finished thread, it gets put back on the queue with the highest priority. If a there is no thread waiting to join with the finished thread, the return value stays in memory for later when another thread joins.

Mutexes

When a mutex is initialized, a queue is allocated. When a thread tries to lock a locked mutex, it is placed in the mutex's queue to be later run when the mutex is unlocked. When a thread unlocks a mutex, the queue is checked. if there are threads waiting, the lock is passed on. If no threads are waiting, the mutex simply unlocks.

Extra Credit A

Every time a thread tries to lock a locked mutex, the priority of the thread is compared with the priority of the thread that currently has the lock. If the priority of the new thread is higher than the old one, the old thread inherits the priority of the new thread. When a mutex is being unlocked and the lock is being passed on to a waiting thread, the waiting thread inherits the priority of the unlocking thread.

Benchmark

pthread:

```
-sh-4.2$ ./parallelCal 6
running time: 667 micro-seconds
sum is: 83842816
verified sum is: 83842816

-sh-4.2$ ./parallelCal 1
running time: 2097 micro-seconds
sum is: 83842816
verified sum is: 83842816
-sh-4.2$
```

my_pthread:

```
-sh-4.2$ ./parallelCal 6
running time: 2098 micro-seconds
sum is: 83842816
verified sum is: 83842816

-sh-4.2$ ./parallelCal 1
running time: 2099 micro-seconds
sum is: 83842816
verified sum is: 83842816
```

The Unix pthread library takes advantage of kernel level thread allowing for true parallelization. That is why the pthread library with 6 threads goes so fast. My_pthread library runs basically the same time with 1 or 6 threads or 1 thread in the pthread library. This is because it is still running on the same core.