Jon Cobi Delfin

Project 1 Report

****Unfortunately I was not able to finish the program due to issues with debugging**

<u>Functions:</u>

my_pthread_create - Every call to my_pthread_create would create and initialize a new tcb. If this was the first call to my_pthread_create, then it would also initialize a runQueue, blockList, and doneList, as well as initializing the scheduler context, sigaction, sigmask, and itimer Each tcb would then be pushed into the runQueue after creation.

my_pthread_yield - The way I designed the scheduler and the run queue was that so the end of each queue would be the currently running context. When this function is called the current running thread would be given the lowest priority and then pushed back into the runQueue;

my_pthread_exit - Since the current running thread was always the last entry in the runQueue, all I simply had to do was dequeue the thread and free it.

my_pthread_join - This function required that a specific thread passed in was to be joined. This meant that the runQueue, blockList, and doneList. If the target thread was already finished, then all I had to do was look for it in the doneList and free the memory. If the target thread was blocked or queued in runQueue, then I had to wait for it in order for the current thread to run. In this case, I made a call to my_pthread_yield, where it would force the current thread to wait for the target to finish.

my_pthread_mutex_init - the mutex I designed only had one struct member named locked. It wasn't necessary to keep track of the mutex id because the user should be locking/unlocking mutexes in total order to avoid deadlock.

my_pthread_mutex_lock - Initially I would have liked to use the test_and_set functions, however they were not available. Instead I used sigprocmask(SIG_BLOCK,...) to block

SIGALRM, preventing the context switch to the scheduler as well as setting the mutex->locked = 1. This guarantees that the current thread gets access to the critical section. If threads try to lock the already locked mutex, they get marked as blocked and wait in the blockList until the mutex is unlocked.

my_pthread_mutex_unlock - This function uses sigprocmask(SIG_UNBLOCK,...) to receive signals after it has set mutex->locked = 0. In addition, calls to this function will remove all threads that were currently blocked and place them back in to the runQueue, where they will have to race again to gain control of the mutex.

my_pthread_mutex_destroy - Since no memory was allocated for the mutex, it simply unlocks the mutex and set the pointer to it equal to null.

sched_stcf - This scheduling algorithm is simply a priority queue. Every time a SIGALRM is raised, the thread at the end of the queue gets to run. When it runs, it gets its priority number incremented, which make it less likely to run. This ensures that threads with a lower priority number get to run

sched_mlfq - unfortunately I was not able to implement this in time

Benchmarks + comparisons - N/A