Thomas Burns

tjb234

Project 2 Writeup

I completed the project and was able to run the code with no segmentation faults and with the proper functionalities of all methods that were implemented for the user thread library.

a. I utilized an array for the scheduler that allowed for a maximum of 20 threads, since the project description described the max test case as containing a possible 10 threads to be scheduled and run.
   The linear array allowed for me to traverse through the list of threads, checking the status of them and running the next RUNNABLE thread. When swapping from contexts I made sure to store the state of the current context that was being swapped from while swapping out to another thread's context.

b. The most challenging part of implementing my user-level thread library was trying to understand why my program was producing segmentation faults. When running, my program would correctly swap between the contexts of the threads that were created until it went to swap back to run a thread for a second time. A thread would only run once, and if the scheduler tried to swap back into it the program a second time it would seg fault every time. I tried multiple solutions such as not saving the context or never saving the new main's context and had some success, but the temporary solutions did not work in every case. They were not following the correct logic of the thread scheduler, so I had to keep trying to figure out what was going wrong.

```
//char context_stack[32768];
int stacksize = 32768;

//scheduler[currrentThread].context.uc_stack.ss_sp = context_stack;
scheduler[currentThread].context.uc_stack.ss_sp = malloc(stacksize);
```

   Only after changing the two commented lines out was I able to fix my program. I had been incorrectly allocating the stack for each for each of the contexts of the threads by storing each of the stack pointers in the SAME LOCATION by using a variable to point to the stack. When my scheduler attempted to swap back into the context of a given thread it would seg fault because that stack then had the stack of a different thread.

   This took me five to six hours of troubleshooting to figure out and I'm definitely not mad.

c. I think that the data structure used for the round-robin scheduling could have been more efficient. If I had used a queue to store what the next context to be swapped to, I would only have to ensure that my insertion method was efficient. Then swapping to the next context in the queue would be O(1) by just popping off the top context. The other point of the program that needed to be addressed was clearing out a my_pthread_tcb of a thread that had finished to allow for more threads to be created and run. This could be done rather easily but was not necessary for the specifications of this project, however, so I did not implement this.