Anu Khatri

CSCE 410-500

Professor Da Silva

23 April 2021

P5 – Design Document

queue.H

I created a new file for the queue implementation since I used geeksforgeeks's linked list implementation. I named the struct RQueue for ready queue. I originally tried to write it myself, but it did not make as much sense since I couldn't get the nodes to enqueue/dequeue, so I went the route of using the geeksforgeeks implementation as a base. The only thing that I added was returning a Thread* instead of dequeue() being a void function. This is used later in scheduler functions, so it made more sense to know what was getting dequeued and verifying that is what we wanted dequeued.

scheduler.H

I added an RQueue object for the queue, as well as readyQueueCount used to keep track of the length of the queue. These are both private members of the Scheduler class.

scheduler.C

Constructor

I just initialized readyQueueCount to 0 since there are no threads yet.

yield()

If there is a thread in the queue to dequeue (meaning that the count is > 0), then I dequeue it. I store that thread returned from dequeue, and pass that thread into Thread's dispatch_to() to invoke the context switch code. Lastly, I reduce the readyQueueCount by one because a thread was dequeued at the beginning of the function.

resume()

I add the thread to the ready queue using the enqueue method. I also increment readyQueueCount because a thread was enqueued.

add()

This just called resume() like it was suggested in the scheduler.H file comments

terminate()

Due to the FIFO nature, I decided to not extract and remove the thread from the middle of the queue (which also would've been more difficult since I did not implement a doubly linked list). Therefore, I instead decided to go through the whole queue. I dequeued the first thread from the queue. If it was the thread we wanted removed, then I decreased the readyQueueCount to account for the removal. If it wasn't the thread we were looking for, then I enqueued that thread back into the end of the queue.

<u>thread.C</u>

I had to add the extern for the System Scheduler so it would be usable for this file.

thread_start()

I enabled the interrupts like suggested in the comments

thread_shutdown()

I had the assumption that it was the current thread. I used Thread::CurrentThread() to get that thread, and then I called Scheduler’s terminate for that thread. I called the Scheduler’s yield() function in order to switch the context after the termination/deletion.