Blake Savell, Xavier Christopher

Dr. Xiaoyu Zhang

CS443- Operating Systems

Assignment 1 Report

***Submitted Files:***

* Pcb.h
* Pcbtable.h
* Readyqueue.h
* Readyqueue.cpp

***How to compile/Run the Program:***

* The command that was used: Make
* ./test1 && ./test2
* g++ test1.cpp (or test2.cpp) pcbtable.cpp readyqueue.cpp ⇒ followed by ./a.out

Unfortunately, this method was time consuming, so then the command to

use most of the time would be ./test1 && ./test2

***Results and Runtimes of Test 2:***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Attempt 1:*** | ***Attempt 2:*** | ***Attempt 3:*** | ***Attempt 4:*** | ***Attempt51:*** |
| ***0.0789381 seconds*** | ***0.0714983 seconds*** | ***0.0976058***  ***Seconds*** | ***0.0711768 seconds*** | ***0.0674251 seconds*** |

With those numbers tested, we ended up with an average runtime of 0.07732882 seconds.

***Features Implemented (via Class):***

1. ***PCB:*** The purpose of this class is to process the incoming data, issue an integer priority and assign it a state (i.e. NEW, READY, RUNNING, WAITING)

* For the PCB.h, there are two classes. An Enum class that holds the ProcState, which will be passed to the readyQueue. The second class is the PCB class. In the PCB class, it consists of a PCB constructor, PCB deconstructor, getID (returns the ID), getPriority (returns priority), getState (which is part of the enum ProcState class), setState (gets the ProcState state from the ProcState class), and setPriority (returns the priority).

1. ***PCBTable: Class PCBTable***The purpose of the PCBTable is to set the array of the PCB elements into the PCB table.  
   - getPCB: to return the PCB at an index.

* addPCB: adds new PCB to the table.
* addNewPCB: Adds new PCB to the table, gets pid ID of the new PCB table.

1. ***readyQueue: Class ReadyQueue***

* Here the class ReadyQueue is loaded with member variables PCB\* heaparray (array of pointers), int capacity (which sets the capacity of the heap), and int count (holds the number of elements in the heap). There is a readyQueue constructor and ~readyQueue destructor (take care of memory leaks), addPCB (adds the next ponter to the PCB), removePCB (returns the PCB with the highest priority), int size (returns the number of elements in the queue), and a displayAll function (displays the PCBs in the queue.

***Choice of Data Structure:*** We used a Vector for the PCBTable. We used Heap for readyQueue. We found this combination to be effective because we could use the vector to quickly set the PCBTable, then, since we were restricted from using the Standard Library in readyQueue, we used the heap, which is very effective for setting and removing from a queue.

***Lessons Learned:***

For both Blake and Xavier, using more office hours could have prevented some point loss. We did have some deficiencies in data structure knowledge, but it was very effective to reach out and ask for assistance. Xavier mentioned that he would work on learning the debugger more to better assist in the debugging process. Both Xavier and Blake agreed that going over data structures would help with future assignments.

***References/Resources:***

* CSUSM STEM Center
* Course materials
* <https://stackoverflow.com/>