Assignment #4 - Scheduler Simulator

Grade Weight: 100pts

Assignment Goals:
- Apply concepts of the process management subsystem to simulate fair and efficient scheduling of processes on a CPU
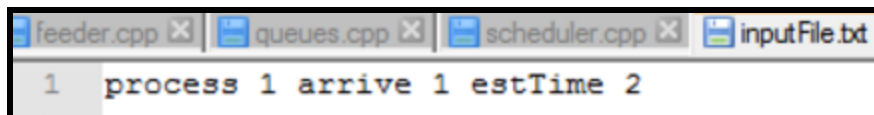
WARNING! All submissions will be scanned for similarity using MOSS software. Your code should <u>NOT</u> be significantly similar to another student, or you will BOTH receive a zero and may be reported to the university for academic dishonesty. It is acceptable to help each other understand the homework, assist a classmate fix a bug, work collaboratively to diagram the solution or draft pseudo-code together. However, sharing significant portions of your code is considered cheating!

Your Task:
- Implement a *simulated* Operating System Scheduler
- Since we are simulating the scheduler in this assignment, it's OK to use C++ to compile your program. *If you curious to why, it's because this allows a little bit more higher-level functionality (abstraction from the hardware) to complete the program. The sample code is written in C++ using the g++ compiler. The program is very similar to C, but with a couple C++ libraries to make it easier to read from the file and print to the screen.*
- The simulated scheduler should implement Round Robin Scheduling
- Assume there is only 1 CPU
- No process should be able to run concurrently for more than one quantum
- If a process runs for more than one quantum and didn't complete, it is put at the end of the ready queue
- When a process returns from a block state, it is also added to the back of the queue.
- To make it a bit easier, if a process blocks or terminates, assume that it happened at the end of a quantum. In other words, no process will ever be preempted in the middle of a quantum.

Test / Input Data:
> **Non-Blocking Processes:** The input data depicts processes arriving and requesting CPU time. For instance, in line #1, we see a process #1 is ready to run at quantum #1 (arrive = 1). It will require two quantums of CPU time to be able to complete (estTime = 2).

> **Blocking Processes:** Imagine some processes will make a system call and therefore block. For instance, in line #2, we see an example. Process #2 arrives at (quantum =

2). It requires 2 quantums of CPU time to complete (estTime = 2) and will block after the first running for one quantum (block **1** 2). Since the kernel-level request takes time to complete (e.g. access data on disk), the system needs to block for a minimal amount of time. In this example, it must block for two quantums (block 1 <u>2</u>).

Sample Code: A significant amount of sample code is provided:
- **feeder.cpp** → reads the input file, simulates the clock interrupts (clock ticks) representing each quantum). Calls the scheduler at each clock tick.
- **queues.cpp** → contains very helpful functions for managing queues
  - **push**: add to the end (tail) of the queue
  - **pop**: remove from head of the queue
  - **remove**: for this assignment, there is a situation in which you would want to move from the middle of the queue (it's like cutting in line). A function is provided to assist you.
- **scheduler.cpp** → this is where most of your effort is needed.
  - Keep track of the state of each process
  - Output changes in process state
  - Force only one process to be running on the CPU at any one time.

SAMPLE CODE & TEST DATA: [Available Here](#)
There are screenshots of example output at the end of this document. Make sure to scroll all the way to the end to see the example output based on the provided input files.

Submission Instructions:
- Submit the code and Makefile on Blackboard inside a single .tar ball
- Your tar ball should be well organized (type *make clean*) to clear any extra files
- Make sure to test that you are submitting the correct tar ball by extracting it and checking the contents
  - Create a directory: mkdir tmp (so you don't overwrite your existing folder)
  - Copy the tar file into the directory cp filename.tar tmp/
  - Extract the tar file "tar –xvf filename.tar"

Rubric:

Properly Submitted: Code submitted correctly. Contains Makefile and the required programs contained in a single .tar file on Blackboard.	**10 points**
Organization & Error Checking: Code is well organized, contains	**10 points**

reasonable error checking and useful comments	
Functionality (Simple) - Successful test using simple.data input file	**30 points**
Functionality (Simple Block) - Successful test using simpleblock.data input file	**25 points**
Functionality (Big Test) - Successful test using comprehensive test (big.data) input file	**25 points**
TOTAL	**100 Points**

*** Submission must compile using the MakeFile. Submissions that do not compile will receive a grade of a zero.*

Sample output from the three test files is available below:

simple.data

```
2 | RUNNING: 1 runTime: 0
3 | RUNNING: 2 runTime: 0
4 | RUNNING: 3 runTime: 0
5 | RUNNING: 1 runTime: 1
6 | RUNNING: 2 runTime: 1
7 | RUNNING: 3 runTime: 1
8 | RUNNING: 1 runTime: 2
9 | RUNNING: 2 runTime: 2
10 | RUNNING: 3 runTime: 2
11 | RUNNING: 1 runTime: 3
12 | RUNNING: 2 runTime: 3
13 | RUNNING: 3 runTime: 3
14 | RUNNING: 1 runTime: 4
14 | TERMINATED TASK: 1
15 | RUNNING: 2 runTime: 4
15 | TERMINATED TASK: 2
16 | RUNNING: 3 runTime: 4
16 | TERMINATED TASK: 3
---------------------
Total Processing Time: 16
Average Wait Time: 7
Max Wait Time: 14
```

big.data

Sample output available here

simpleblock.data

```
2 | RUNNING: 1 runTime: 0
3 | RUNNING: 2 runTime: 0
3 | BLOCK TASK: 2
BLOCK WAIT TIME: 2
4 | RUNNING: 3 runTime: 0
5 | RUNNING: 4 runTime: 0
ADDING TO READYQ: 2
6 | RUNNING: 1 runTime: 1
7 | RUNNING: 3 runTime: 1
8 | RUNNING: 4 runTime: 1
9 | RUNNING: 2 runTime: 1
10 | RUNNING: 1 runTime: 2
11 | RUNNING: 3 runTime: 2
12 | RUNNING: 4 runTime: 2
13 | RUNNING: 2 runTime: 2
14 | RUNNING: 1 runTime: 3
15 | RUNNING: 3 runTime: 3
16 | RUNNING: 4 runTime: 3
17 | RUNNING: 2 runTime: 3
18 | RUNNING: 1 runTime: 4
18 | TERMINATED TASK: 1
19 | RUNNING: 3 runTime: 4
19 | TERMINATED TASK: 3
20 | RUNNING: 4 runTime: 4
20 | TERMINATED TASK: 4
21 | RUNNING: 2 runTime: 4
21 | TERMINATED TASK: 2
---------------------
Total Processing Time: 21
Average Wait Time: 8.7
Max Wait Time: 18
```