Zach Stence
3/29/2019
CS 4328

Compile and Run Instructions (taken directly from README.md)
First, compile the main class

```
$ g++ main.cpp
```

To simulate one scheduling algorithm with a specified arrival rate, service time and quantum length (only used in RR), run the following command choosing from the following algorithms
1. First Come First Serve (FCFS)
2. Shortest Remaining Time First (SRTF)
3. Highest Response Ratio Next (HRRN)
4. Round Robin (RR) with specified quantum length

```
$ ./a.out <scheduler> <arrival rate> <service time> <quantum length>
```

For example:

```
$ ./a.out 2 10 0.06 0.01
 Avg. Turnaround Time :  0.094
           Throughput :  9.933
        Avg. CPU Util :  0.593
  Avg. in Ready Queue :  0.344
```

To simulate all the scheduling algorithms with arrival rates from 1-30 processes/second, a service time of 0.06 seconds, and quantum lengths of 0.01 and 0.2 seconds (for round robin), simply provide '-1' as the scheduler choice

```
$ ./a.out -1
Simulating FCFS...done
Simulating SRTF...done
Simulating HRRN...done
Simulating RR(0.01)...done
Simulating RR(0.2)...done
Finished all simulations.
```

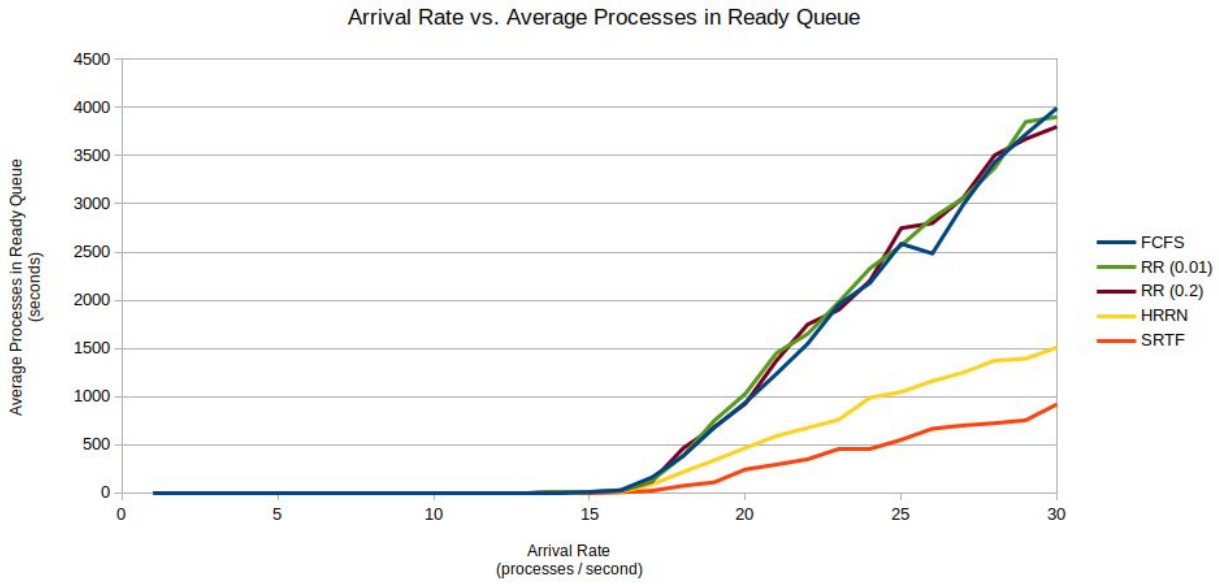
The results will be saved in 5 CSV files, each named for its corresponding scheduler. The CSV files have 4 columns (arrival rate, average turnaround time, throughput, and average processes in the ready queue), each holding values for a different iteration. These files can then easily be imported into a spreadsheet application to make graphs from the CSV results like I have included in this report and on my GitHub repo.

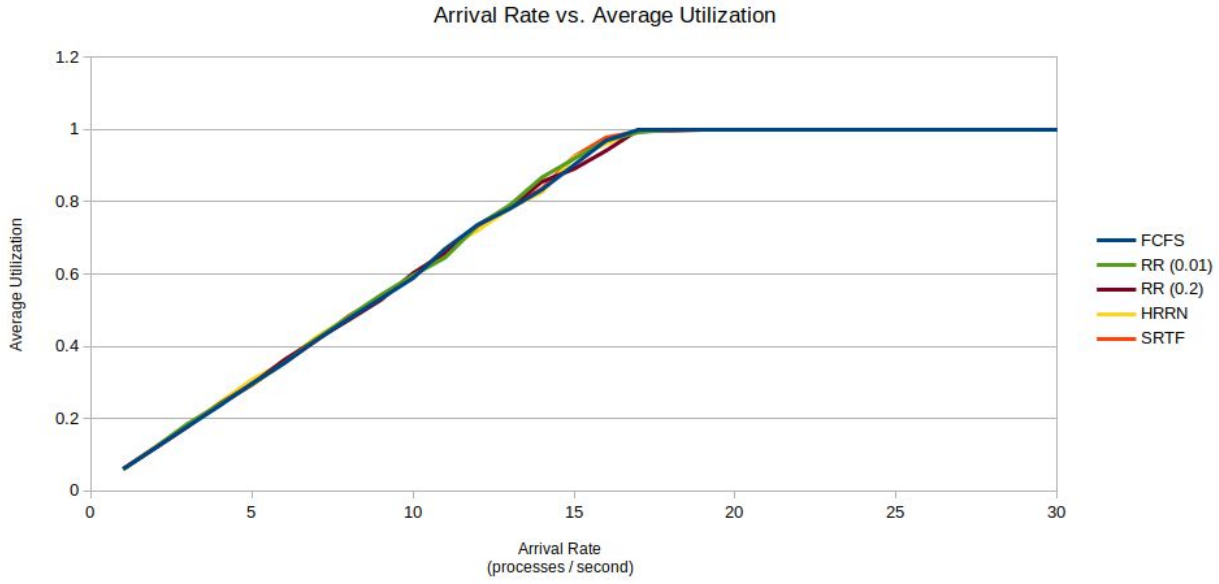
Observations

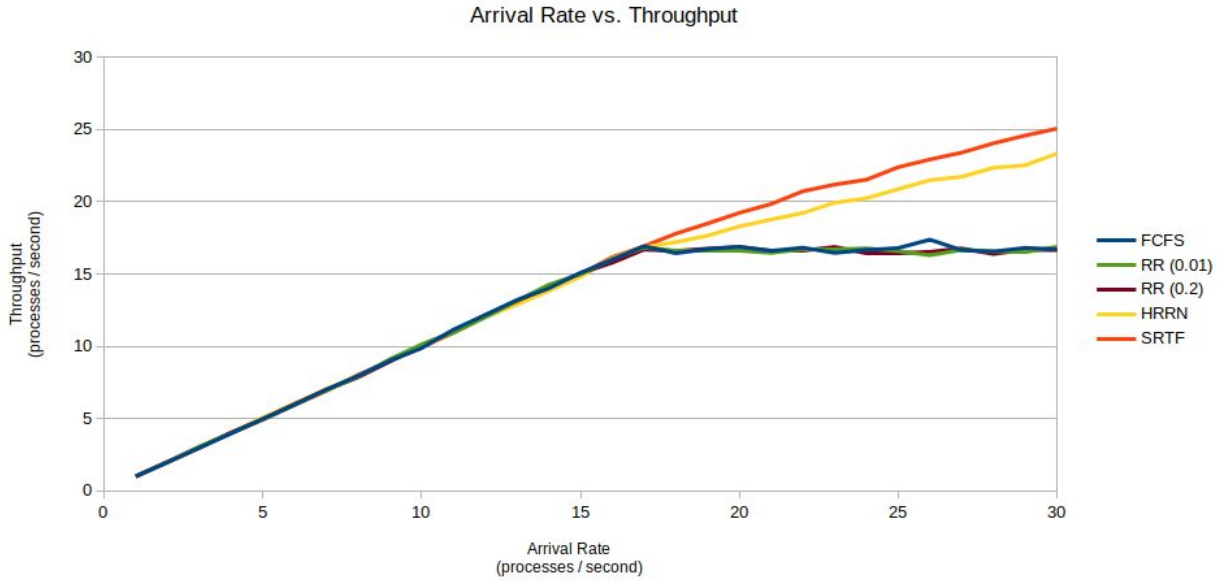
From the graphs I generated I noticed that overall, the schedulers performance was ranked as follows (best to worst) SRTF, HRRN, RR, FCFS. This makes sense when considering what we learned in class. In class you mentioned that as the quantum length for RR increased, its behavior approached that of FCFS. I definitely see this happening with a quantum of 0.2 seconds, but also for 0.01 seconds which I didn't expect. I'm not sure how the performance of 0.01 should be different, perhaps there's a bug in my code. Also, on my graph for CPU utilization, all schedulers followed roughly the same curve which I did not expect. However, this makes sense because regardless of the scheduling choices made, there is still the same rate of processes coming in and going out, which should result in a similar utilization. Aside from that, I noticed that the throughput of SRTF and HRRN doesn't cap when the CPU reaches 100% utilization unlike FCFS and RR. This is because SRTF and HRRN put processes with a shorter service time at a higher priority. Lastly, I noticed that the curves on the average turnaround time graph appear to be logarithmic, meaning that if we continue to increase the arrival rate, the average turnaround time shouldn't blow up to huge numbers. But, the curves on the throughput and average processes in the ready queue graphs appear to be linear.

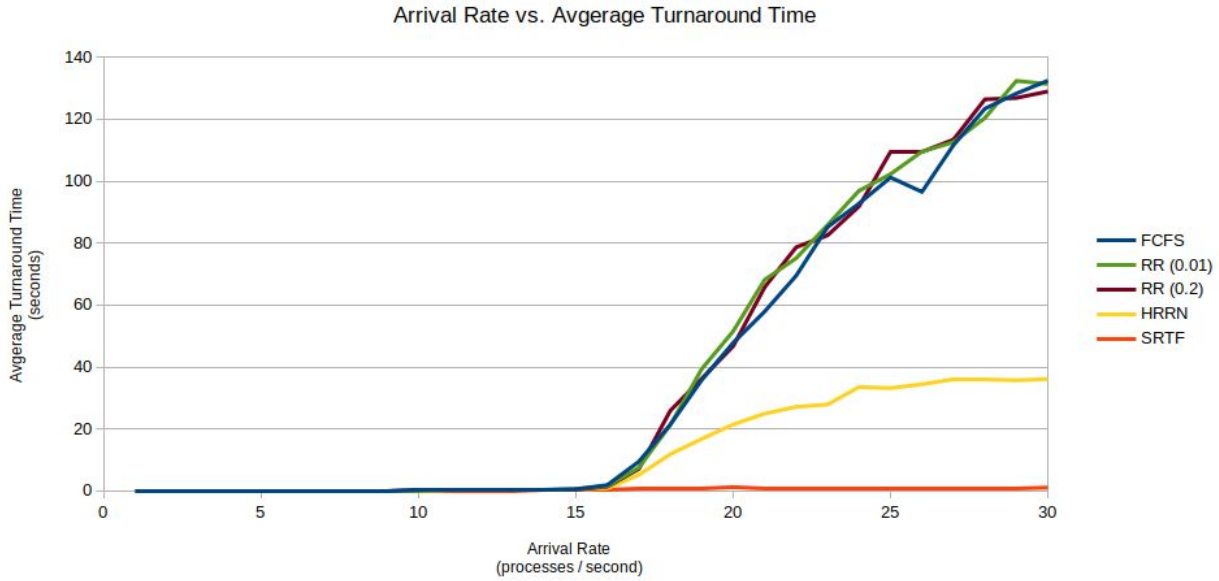
This was a very interesting project and I enjoyed every part of it! I have uploaded the code to my GitHub (https://github.com/zachstence/SchedulerSimulator) as well as the output CSV files, graphs I made, and a proper readme. I plan on adding more features to my code in the future.

Graphs


Arrival Rate vs. Average Processes in Ready Queue
Average Processes in Ready Queue (seconds)
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
5
10
15
20
25
30
Arrival Rate (processes / second)
FCFS
RR (0.01)
RR (0.2)
HRRN
SRTF


Arrival Rate vs. Average Utilization
Average Utilization
1.2
1
0.8
0.6
0.4
0.2
0
0
5
10
15
20
25
30
Arrival Rate (processes / second)
FCFS
RR (0.01)
RR (0.2)
HRRN
SRTF


Arrival Rate vs. Throughput
Throughput
(processes / second)
30
25
20
15
10
5
0
0
5
10
15
20
25
30
Arrival Rate
(processes / second)
FCFS
RR (0.01)
RR (0.2)
HRRN
SRTF


Arrival Rate vs. Avgerage Turnaround Time
Avgerage Turnaround Time
(seconds)
140
120
100
80
60
40
20
0
0
5
10
15
20
25
30
Arrival Rate
(processes / second)
FCFS
RR (0.01)
RR (0.2)
HRRN
SRTF

My Code

The main.cpp file contains a main function that parses the input from the command line. You can run it as specified in the project assignment, or if you pass '-1' for the scheduler choice it will run all the schedulers for various lambda values and save the output in CSV format.

My code is likely a little different than most other submissions so I'd like to explain it a little. I first began with the FCFS algorithm and finished it pretty quick with no issues which allowed me to figure out all the data structures I needed and how to calculate all the statistics. Next I copied the general structure to SRTF and implemented that algorithm. Upon moving on to HRRN I noticed that my code was very similar to what I wrote for FCFS so I decided to combine them into one function that takes a Comparator as a template parameter to control the priority scheme. The only difference between FCFS and HRRN is that HRRN uses dynamic priorities, so I made that small change and combined them into one function with a boolean controlling whether or not to use dynamic priorities. Then I noticed that SRTF was similar as well, the only difference being that it uses preemption, so I decided to combine everything into one function with an additional boolean for whether or not to use preemption. This allows for (as far as I know) any priority based scheduler to be tested by adding a new comparator and setting the proper values for dynamic priority and preemption.

As far as the efficiency and time complexity of my code, I'm pretty proud of everything except for the resorting of the ready queue required for schedulers with dynamic priorities (HRRN). Since I used a set as the underlying data structure for my ReadyQueue class, the only way to resort it for the dynamic priorities was to re-add all the elements to a new set which is O(n) or worse. If I had some more time, I would have created my own ReadyQueue and EventQueue classes completely from scratch with proper insertion, deletion and sort methods with the best time complexities possible.