Simulation of CPU Scheduling Algorithms Report

Brief overview of the design and implementation:

For the design of First-Come First-Served (FCFS) [non-preemptive] algorithm the implementation was that when an event1 was called in the switch statement it called the FCFS method. In this method it made a new event and process the same as the one that was put into the parameter portion of the function. Except for that it created a new enter_time for the event that was its original enter time plus the remaining service time as the new enter time. Also the event type was changed from 1 to 2 to indicate a completed event. After the new_event is created and connected to the processes it is used as an input for the schedule_event method. This method organizes the new event into the event linked list based on their enter_time.

For the design of the Round Robin algorithm when an event type of 1 is the head node it calls the RR function. Inside of which checks whether the remaining time onthe process is less than the quantum number and if it is then it sets the remaining time to 0 and sets a new event equal to it with a new arrival time of its current one plus the difference between the quantum number minus the remaining time. If the remaining time is greater than the quantum number then the new event created is the same as the last event except for that its remaining service time is subtracted by the quantum number.

For the design of the Shortest Remaining Time First our approach was to each time an event of type 1 was the head node it would call the SRTF function as the parameter in it. In this function we made a new event and processes euall to that of the parameter. And then called the scedule_readyQue with the new_event as the paramter. This function orders the ready queue based on the remaining time of the process from shortest to longest. After that the function SRTF_Helper is called which, based on the condition, subtracts from the headnode of the readyqueue based on the time difference of the current clock and the next jump to clock. Conditions are checked to make sure that it stays the smallest in the linked list. After that an event and process is made and the function schedule_readyQue is called with the event.

For the design of getting the average turnaround time for each of the algorithms we collected the data from when the events enter the type 2 event where they have their data collcted. In this instance we got the result of subtracting the finishing time against the original entry time of the event and process.

For the design of collecting the the total throughput for the algorithms
We took the total amount of time to finish through the linked list and then divided that against the total amount of processes of 10000.

For the design for the average number of processes per algorithm we got the total amount of processes per iteration of the linked list. We have a counter for the

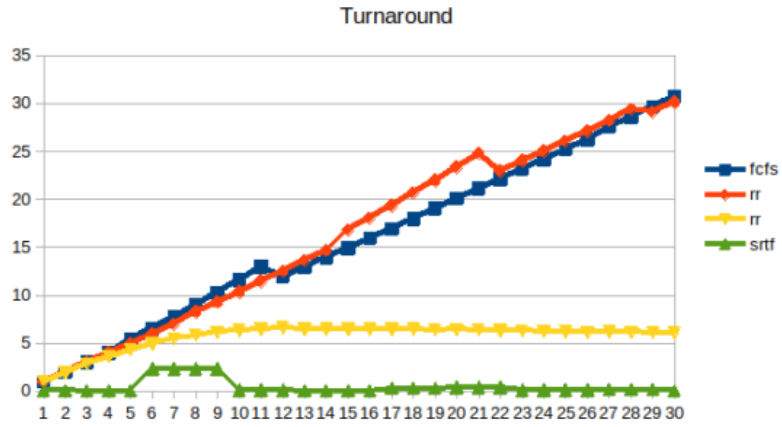
number of iterations and at the end to get the average we just divided the total amount over the amount of iterations.

Instructions on how to compile Program: To compile my program on the linux servers it requires these lines of commands:

1. Compile the .cpp file with "g++ -o program2 program2.cpp -std=c++11”
2. Run the .bat with "./O.S._Program2_Batch_File.bat"

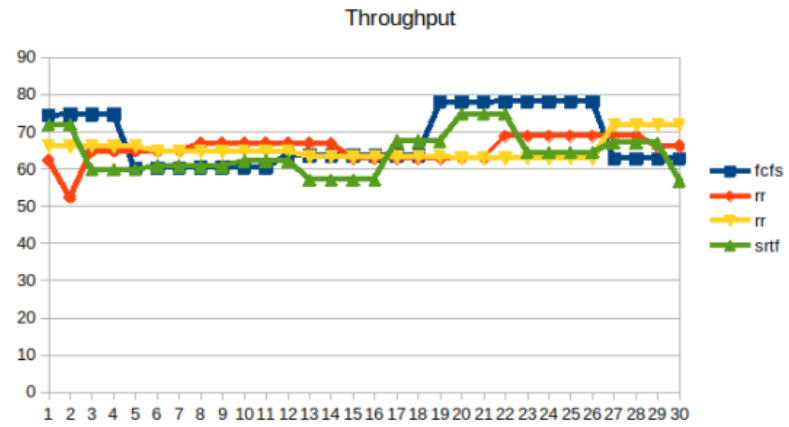
Results of the experiments and their interpretation:

The average turnaround time plot:



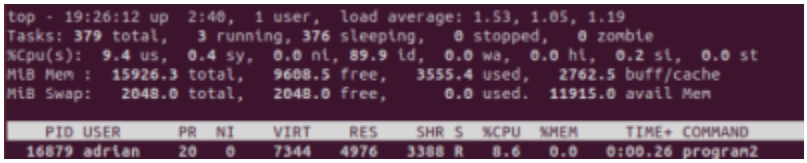
In the average turnaround in first come first and both of the RR are close in results up until the 15 lamda point. They both have a continuous increase over lamda. Where the lower RR has an increased turnaround time due to that it is going to have more processes in the queue.

The total throughput (number of processes done per unit time):



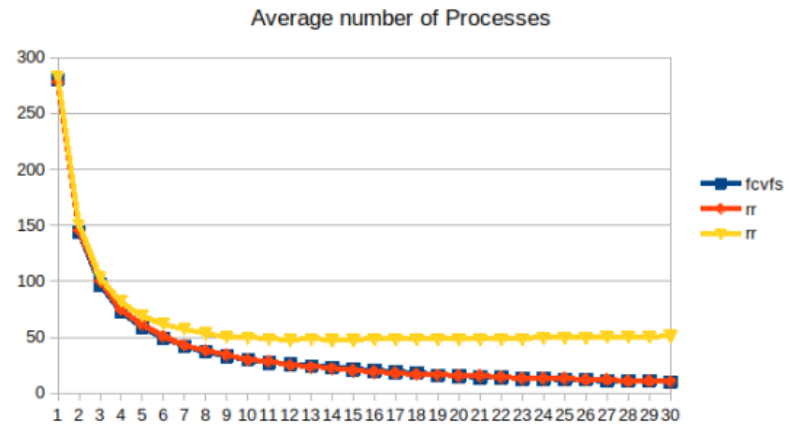
In the graph above all three algorithm types including both round robin types are very close together in results. The First come first serve has the highest throughput at the start but then slows down and then increases near the end. The shortest remaining time first has the second lowest throughput at the start but over the increased lamda the throughput increases. The lower round robin increases throughput over increased lamda. The .2 round robin has a continuous throughput until about the 26th lamda where it starts to increase.

The CPU utilization: 8.6%



In the table we show that while running the algorithms we had on average a 8.6% CPU utilization.

The average number of processes (queue length) in the ready queue:



In the above graph for the algorithms for average number of processes they all level off at the critical point. Where the .01 round robin has the highest average number of processes at around 50. This is due to that in the .01 round robins linked list gets backed up with processes. Due to the small quantum number in comparison to the average burst time of the processes. And the rest of the algorithms are very similar and decrease as lamda increase to 30 near 0.