CS4328: Project #1

Due on Mar, 29, 2019 at 11:59PM

Jamal Rasool

In Collaboration with:

Kristof York

Jamal Rasool
Friday, March 29, 2019

Report.

1. Summary
    a. Within this lab we explored how the different discrete-time event simulators work and play out, with going through and implementing a linked list implementation of the project, we were able to calculate the correct data for the different algorithms. The way we designed and attacked the project was by first creating a way for the code base to accept arguments from the command line, in which then will initialize the program with default values and provided values. Based upon that data and the scheduler that the user / script picks the program will execute the algorithm that is requested. Based upon the data within the algorithm the following functions are called located within the code over view section, in which calculates the average data, throughput, utilization, and the number of processes in the system. These calculations were improvised versions of the homework assignments that we have completed in the past, to fit the code base that we had set forward. Overall the project has been a very demanding project in which took many weeks to get just right to obtain the most accurate results.

2. Readme
    a. In order to run the code, the following commands in the image below will need to be run in order to achieve the correct result. Note, the code should be able to work on C++11 without any issues, and the make file is already pre-configured to handle the program. Also, if there is any issues with running the script.sh file, then you may need to run chmod +x script.sh in order to make it an executable to run. Once you run the script the output of the data will be located within the report.txt file, which displays the info neatly over all of the data that was requested within the project.

3. Code Overview

Jamal Rasool
Friday, March 29, 2019

a. Get Average Turn Around Time Function

```
float getAvgTurnaroundTime() {
   float totTurnaroundTime = 0.0;
   float totalServicetime = 0.0;
   int count = 0;
   int count2 = 0;
   procListNode* pIt = pHead;
   while(pIt->pNext != NULL) {
      // tally up the turnaround times
      if(pIt->finishTime == 0) {
         //cout << "nope" << pIt->finishTime << endl;
         count2++;
      }
      else {
         totTurnaroundTime += (pIt->finishTime - pIt->arrivalTime);
         count++;
      }
      totalServicetime += pIt->serviceTime;
      pIt = pIt->pNext;
   }
   return (totTurnaroundTime / stopCond);
}
```

b. Get Total Throughput

```
float getTotalThroughput( ) {
   procListNode* pIt = pHead;
   float finTime = 0.0;
   int count = 0;

   while(pIt->pNext != NULL) {
      // get the final timestamp
      if(pIt->finishTime == 0) {
         count++;
      }
      else{
         finTime = pIt->finishTime;
      }
      pIt = pIt->pNext;
   }
   cout << "totalTime: " << finTime << endl;
   return ((float)stopCond / finTime);
}
```

c. Get CPU Utilization

```
float getCpuUtil( ) {
   procListNode* pIt = pHead;
   float busyTime = 0.0;
   float finTime = 0.0;
   int count = 0;
   while(pIt->pNext != NULL) {
     if(pIt->finishTime == 0){
        count++;
     }
     else {
       busyTime += pIt->serviceTime;
       finTime = pIt->finishTime;
     }
     pIt = pIt->pNext;
   }
cout << "Busy Time: " << busyTime << endl;
cout << "finTime: " << finTime << endl;
cout << "number of 0 finish time's in CPU util: " << count << endl;
   return (busyTime / finTime);
}
```

d. Get Average Number Of Process In Queue

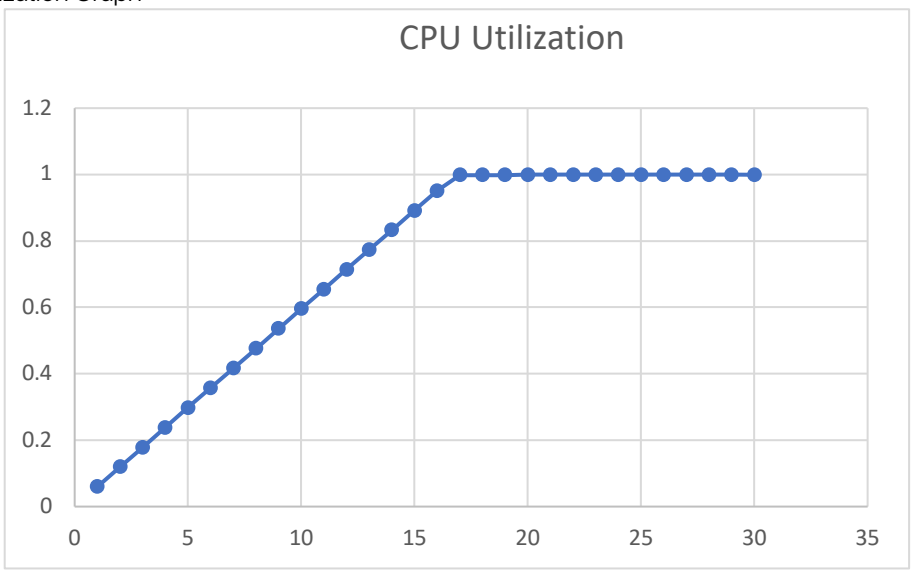
Jamal Rasool
Friday, March 29, 2019

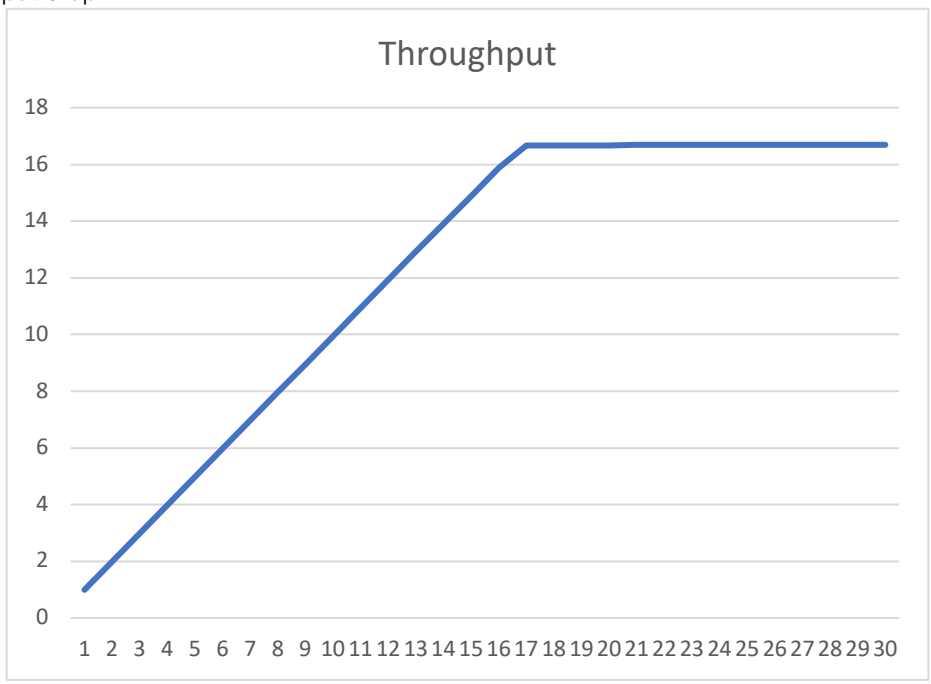
```
float getAvgNumProcInQ() {
   // identify the final second of processing (timeN)
   // as it would appear on a seconds-based timeline
   float timeNmin1 = 0.0;
   int count = 0;
   procListNode* pIt = pHead;
   while(pIt->pNext != NULL) {
      if(pIt->finishTime == 0) {
        count++;
      }
      else {
        timeNmin1 = pIt->finishTime;
      }
      pIt = pIt->pNext;
   }
   int timeN = static_cast<int>(timeNmin1) + 1;

   // tally up the total processes in the ready queue
   // for each second of the seconds-based timeline
   pIt = pHead;
   int time = 0;;
   int numProcsInQ = 0;
   for(time = 0; time < timeN; time++) {
      while(pIt->finishTime != 0) {
         if((pIt->arrivalTime < time && pIt->startTime > time) ||
            (pIt->arrivalTime > time && pIt->arrivalTime < (time + 1))) {
            numProcsInQ ++;
         }
         pIt = pIt->pNext;
     }
     pIt = pHead;
   }

   return ((float)numProcsInQ / timeN);
}
```

Jamal Rasool
Friday, March 29, 2019

Graphs Data.

Shows data regarding each of the schedulers over the 150 different runs of the program itself, notated in the following order of FCFS, SRTF, HRRN, and then RR are shown below.

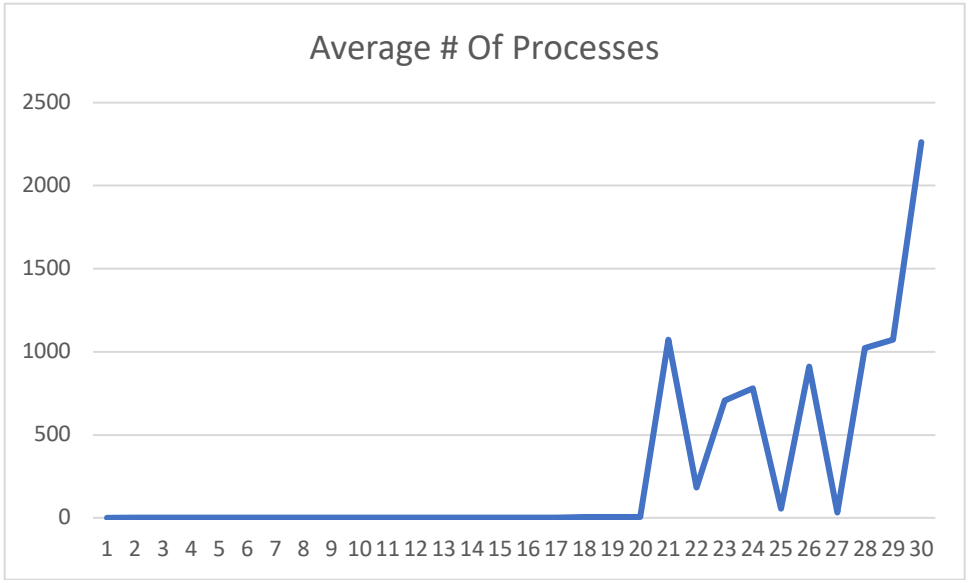
1. First Come First Serve Scheduler (FCFS)
    a. CPU Utilization Graph
    i. 

    b. Throughput Graph
    i. 

    c. Average Number Of Processes

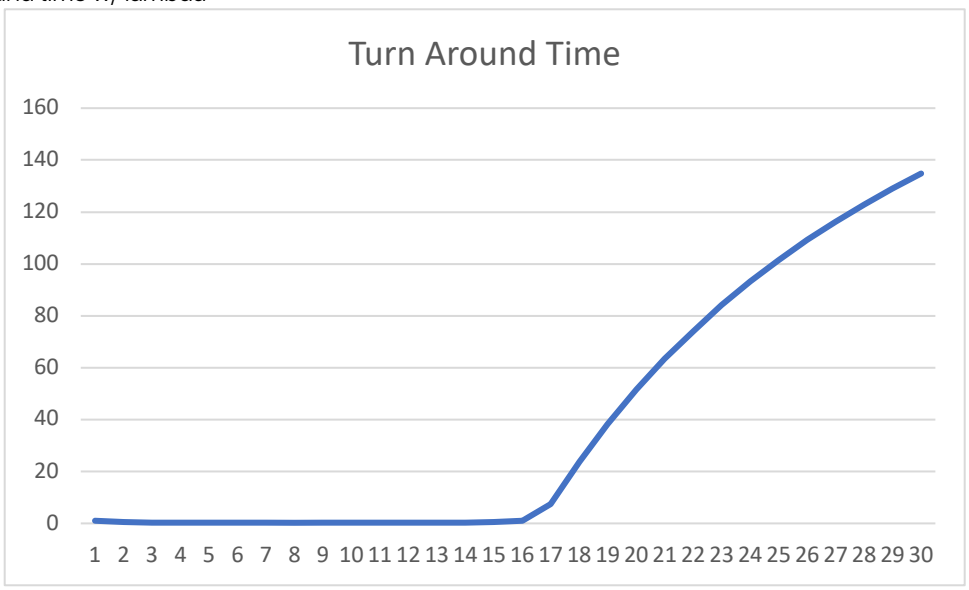
Jamal Rasool
Friday, March 29, 2019

i.



d. Turnaround time w/ lambda

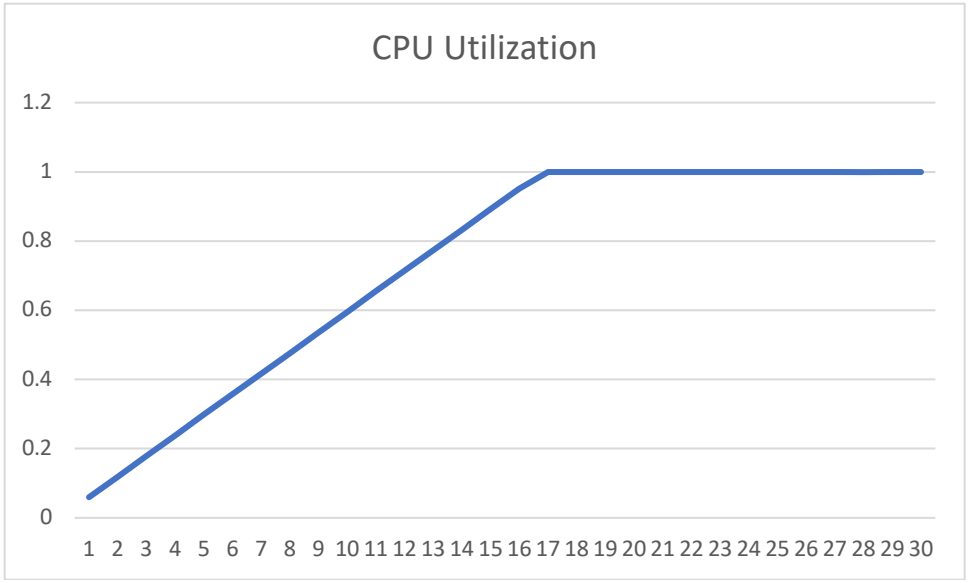
i.



2. Shortest Remaining Time First (SRTF)
   a. CPU Utilization Graph

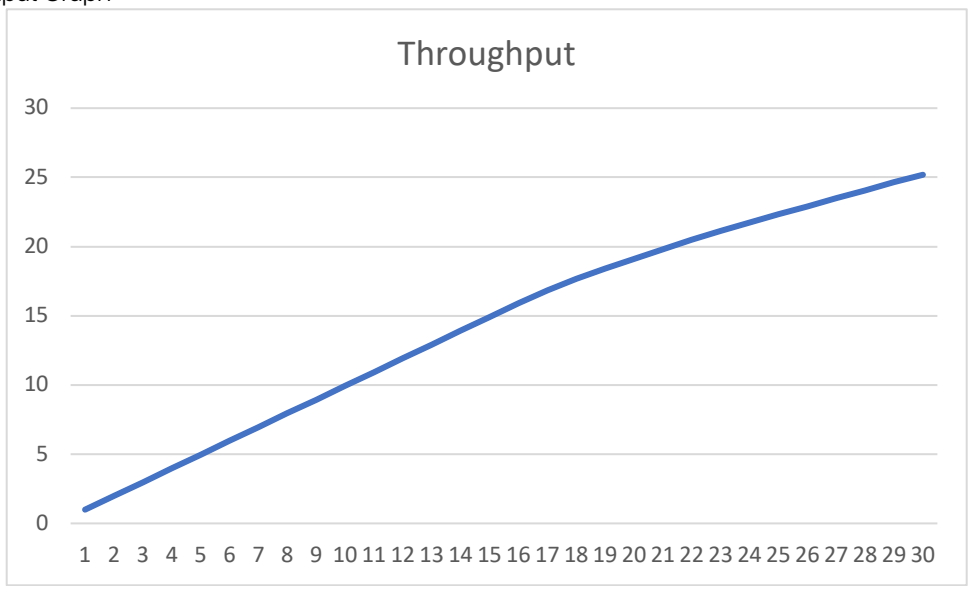
Jamal Rasool
Friday, March 29, 2019

i.

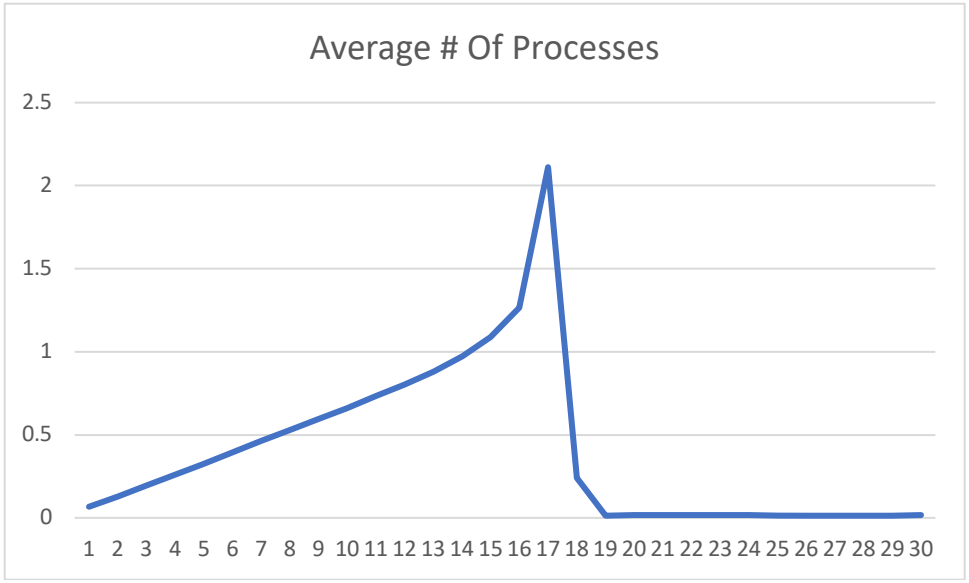


b. Throughput Graph

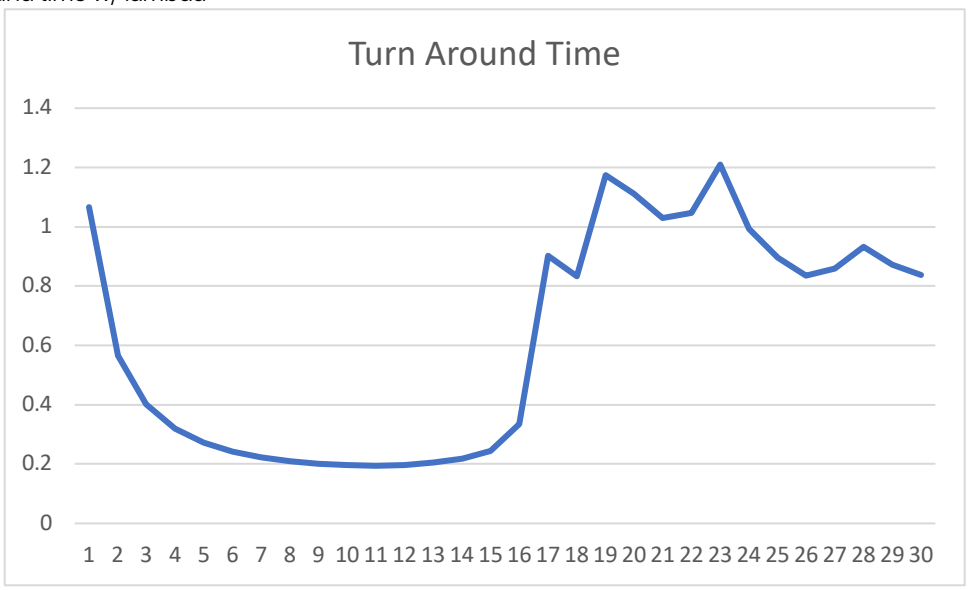
i.



c. Average Number Of Processes

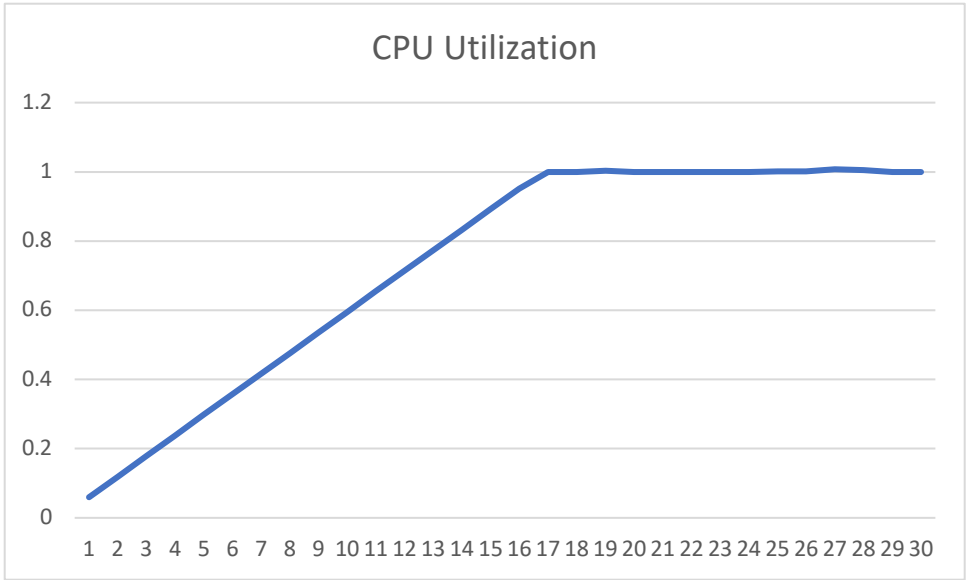
i. 

d. Turnaround time w/ lambda

i. 

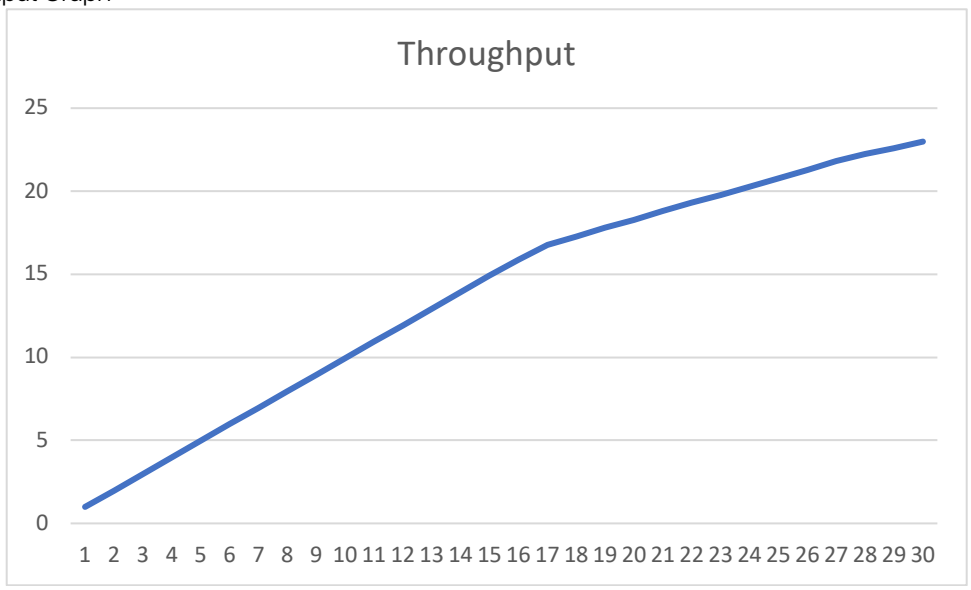
3. Highest Response Ratio Next (HRRN)
   a. CPU Utilization Graph

i.



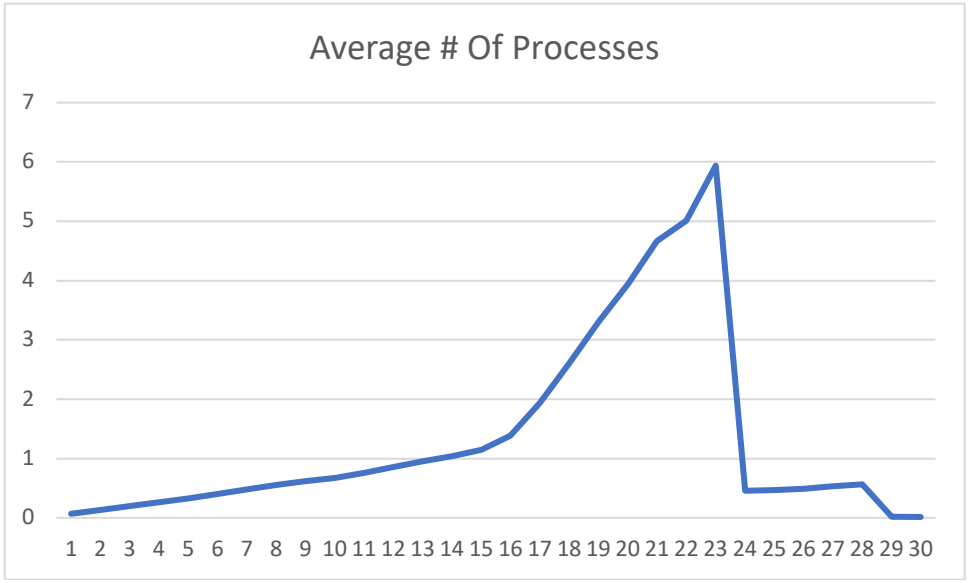
b. Throughput Graph

i.



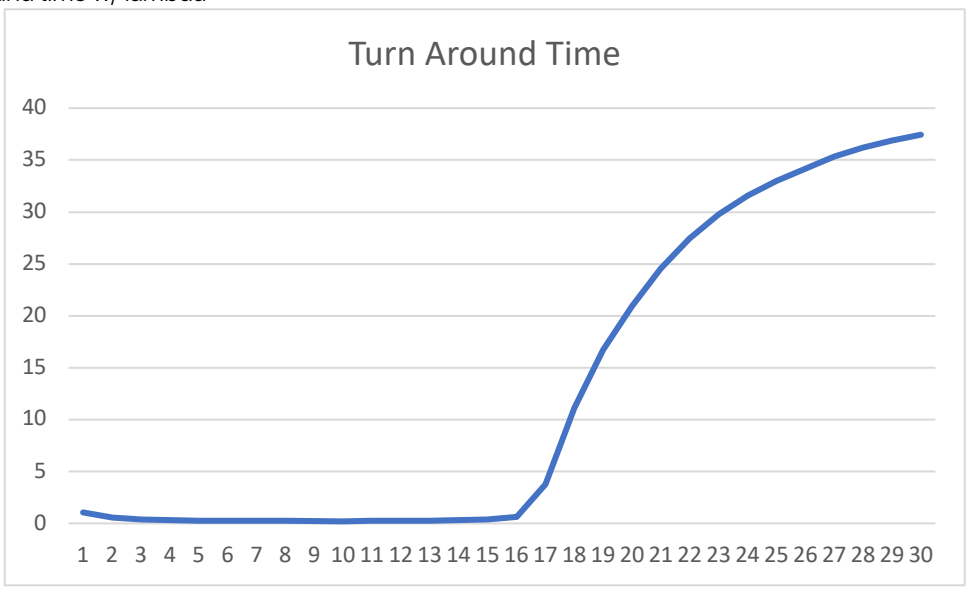
c. Average Number Of Processes

i.



d. Turnaround time w/ lambda

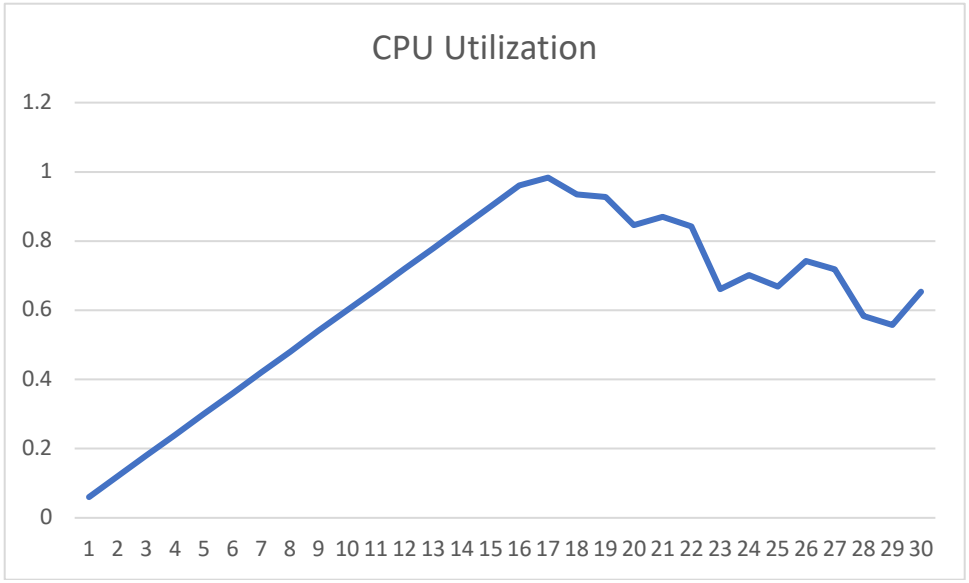
i.



4. Round Robin (RR) with Q=0.01
   a. CPU Utilization Graph

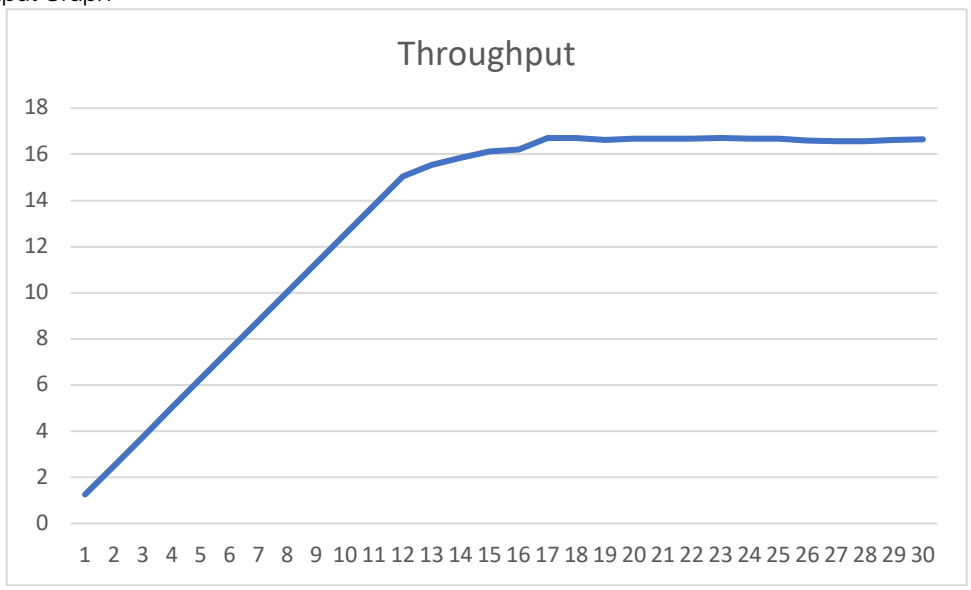
Jamal Rasool
Friday, March 29, 2019

i.



b. Throughput Graph

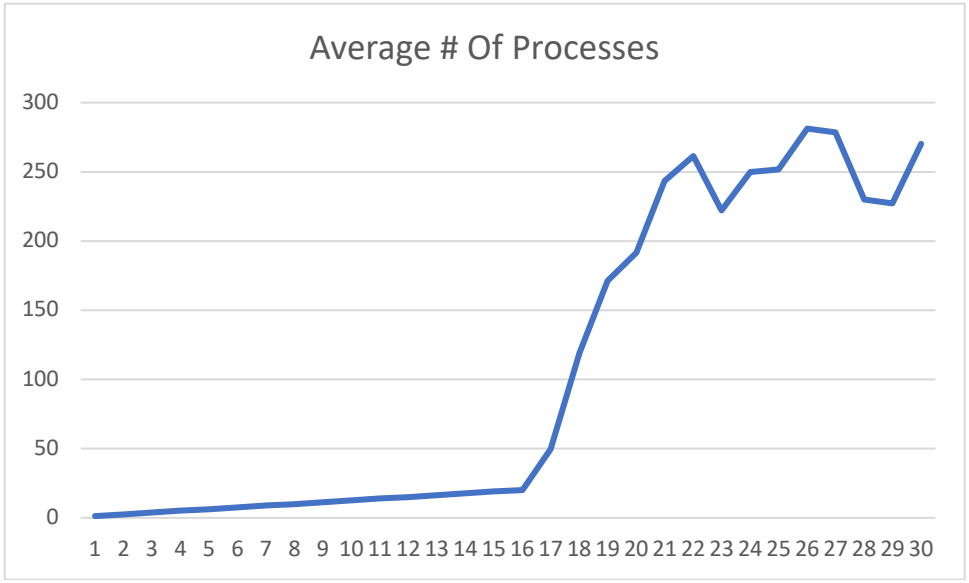
i.



c. Average Number Of Processes

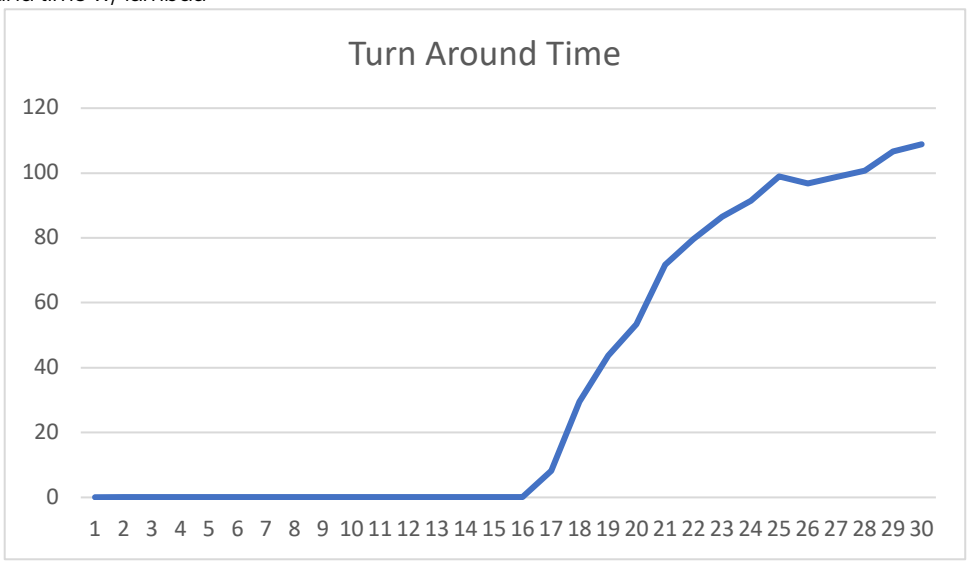
Jamal Rasool
Friday, March 29, 2019

i.



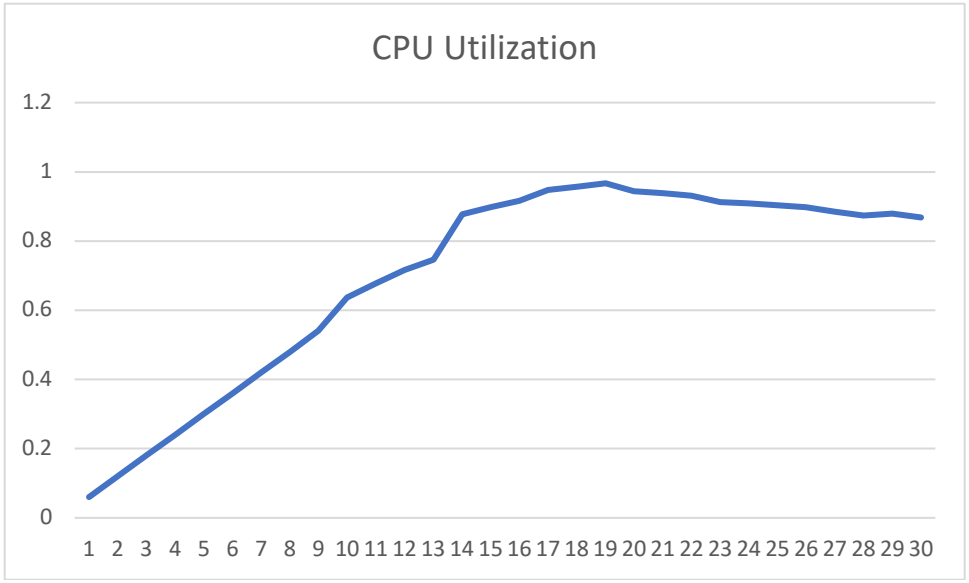
d. Turnaround time w/ lambda

i.



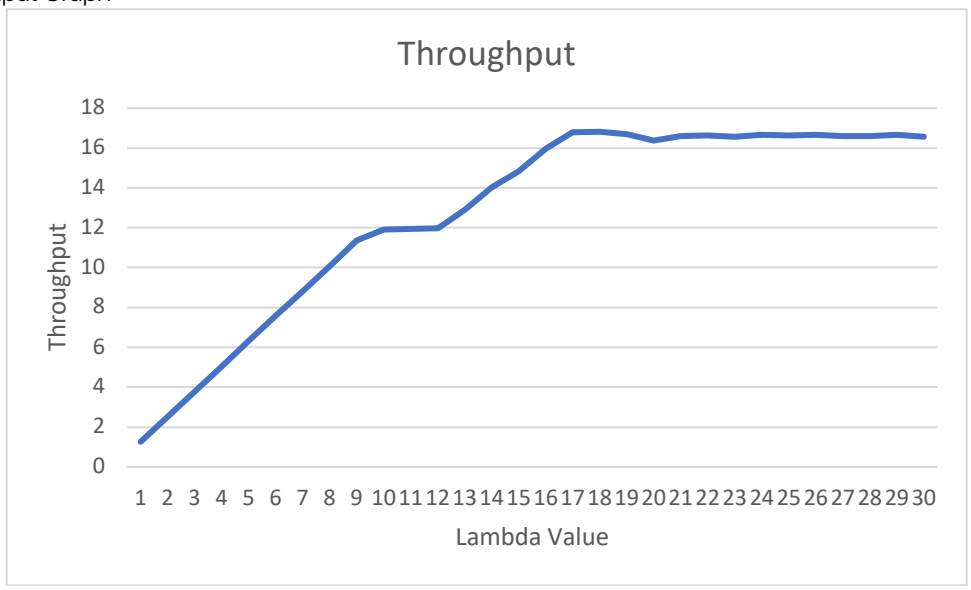
5. Round Robin (RR) with Q=0.2
   a. CPU Utilization Graph

i.



b. Throughput Graph

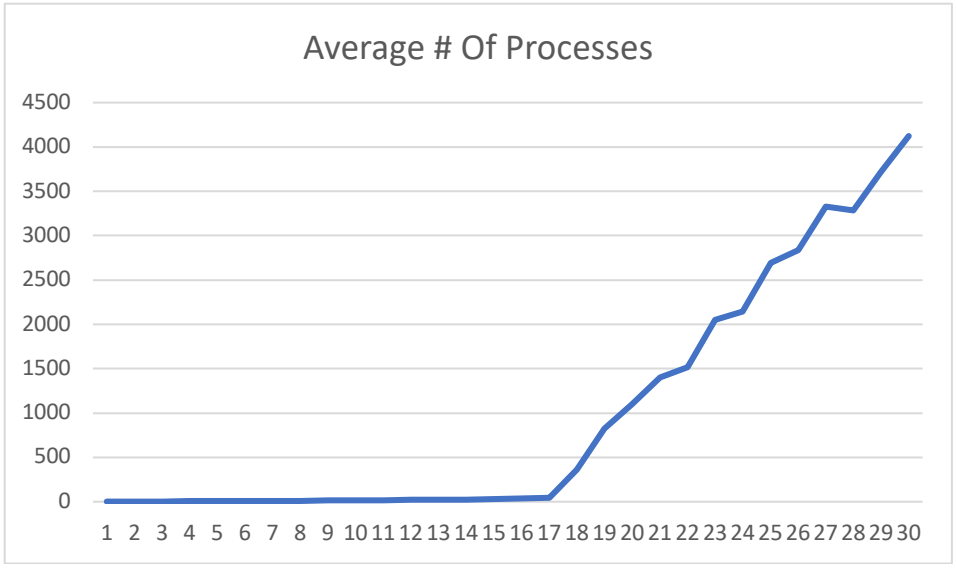
i.



c. Average Number Of Processes

Jamal Rasool
Friday, March 29, 2019

i.



d. Turnaround time w/ lambda

i.