Monte Carlo Simulation of Scheduling Algorithms.

1. Problem Statement

Write a Monte Carlo simulation of the FCFS, RR q=1, HRRN, and FB q=1 scheduling algorithm in a C/C++ program. One process arrives at each time step. Service times are to be normal distribution with a mean of 10 and a standard deviation of 5 (a minimum service time of one must apply). In order to collect meaningful statistics, perform one thousand experiments with one thousand processes each. Include a discussion of the difficulty of implementing each algorithm along with some notion of the how much overhead it would likely represent during execution. Make a decision as to which of these algorithms you would recommend to you manager based on all of the information that you gleam through this assignment.

See Table 9.5

Submit your report and code here.

To implement short-term scheduling algorithms used by an OS to determine process execution order. The algorithms to be implemented were, First Come First Serve (FCFS), Round Robin (RR), Highest Response Ratio Next (HRRN), and Feedback (FB). These algorithms must be run against a batch of 1000 processes. There should be a 1000 of these simulations with 1000 processes each. The service times for the processes are to be a normal distribution with a mean of 10 and standard deviation of 5, with the minimum service time being 1.

The importance of the project is to learn the concepts of short-term uniprocessor scheduling in an OS when the service time for a process is already known. And to learn the use of a Monte Carlo simulation to test multiple scheduling algorithms over a certain dataset and compare their performance relative to each other.

2. Approach

Language used: C++11

Compiler used: g++

Editor used: micro

Debugging tools used: gdb.

First task was writing the data structures that would aid in the process of simulating a processor's function. The data structures called 'process' and 'simulation' were created so as to hold information for each process and the latter to hold information for each batch of 1000 processes simulated.

Next task was implementing the scheduling algorithms and testing for correctness. The algorithms were implemented as concurrent threads in the same loop. Thus, we could think of the system as a 4 core-processor. Each of the processors has their own unique scheduling algorithm and have the same processes coming in at the same time. As the core execute these processes based on the scheduling policy the turnaround times for the processes are measured and stored. Data from 1000 such simulations is then aggregated, and average taken before reporting.

Last task was creating the normally distributed dataset required for the Monte Carlo simulation. Using the C++ STL, 'random', and the class contained within it called 'noraml_distribution' a randomly distributed dataset of real numbers is generated. The numbers in this dataset are then rounded to the nearest integer and the values less than 1 are discarded. The distribution is now an approximation of the normal distribution due to the discretization of the values and the removal of approximately 3.59% of the values (values lower than 1).

3. Solution

No major bugs were detected while implementing the algorithms mentioned earlier.

The solution is merely an adaptation of the algorithms described in the course textbook.

To build the code g++ is invoked as follows.

```
$ g++ -std=c++11 -o prog4 main_rds190000.cpp
```

To run the code the following command is executed.

```
$ ./prog4
```

Comparison of the scheduling algorithms

Criteria	FCFS	RR (q=1)	HRRN	FB(q=11)
Ease of implementation	Very easy to implement, uses a simple queue and a process is executed non-preemptively.	Relatively easy to implement, uses a queue but processes are pre-empted after one time interval of processing.	Most complicated implementation. Requires determination of the process with the highest response ratio and execution of	Slightly more complicated than round-robin since there are multiple queues of different priority levels. Processes are executed preemptively and are bumped to a lower

			the process non-preemptively.	queue after one time interval of processing.
Overhead	Minimal, since there is simple queuing without the need to have minimal process data plus there is no requeuing.	Minimal again, since the process data required is minimal and involves requeuing after set time slices.	High overhead. It requires maintain data such as how long the process has been in the queue, the arrival time of the process and calculation of the response ration. This has to been done for each process in the queue.	Medium to high overhead, since it maintains multiple queues. But all the queues are simple and minimal process information is needed for the algorithm to function.
Mean Turnaround Time	4599	6777.58	3405.83	6872.34
Mean Normalized Turnaround Time	642.46	686.305	271.266	686.284

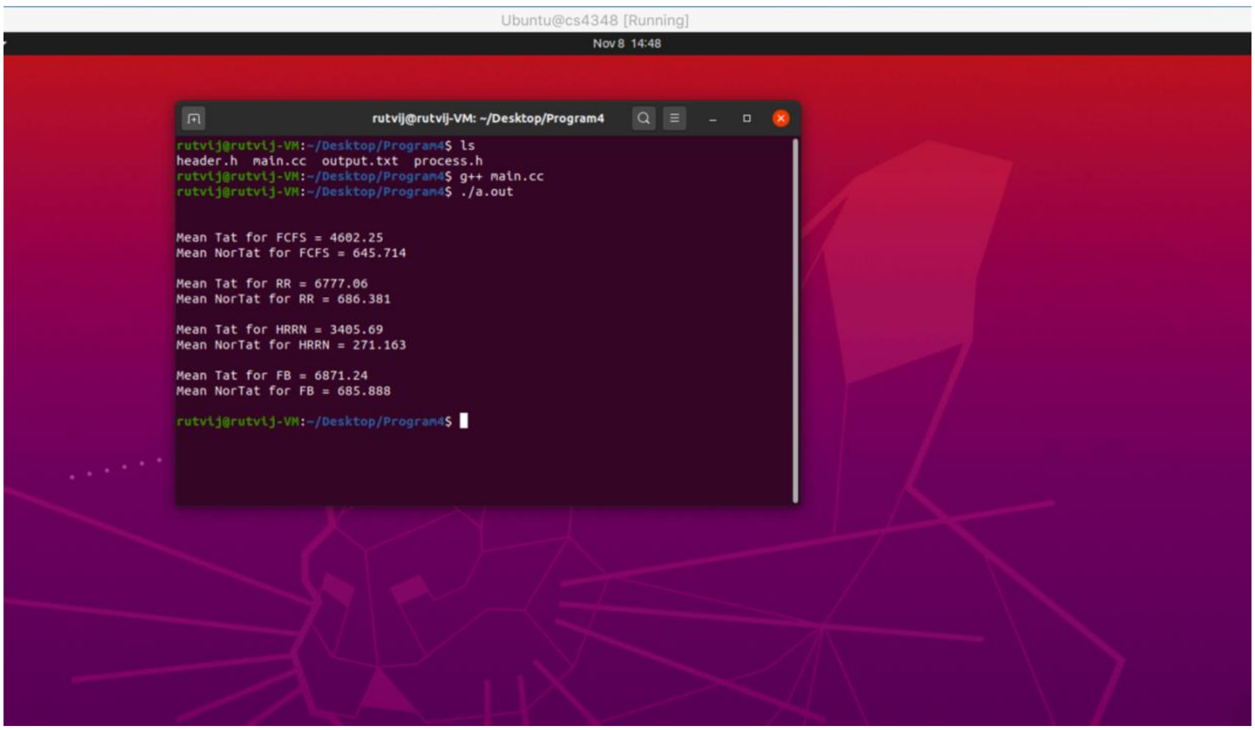
Suggestions

Based on the data obtained from the simulations following suggestions for the choice of the scheduling algorithms can be made.

Performance Priority – ***use HRRN*** algorithm since it has the lowest mean turnaround time (3405.83 time units) and the lowest mean normalized turnaround time of 271.266 which is less than half the value for all other algorithms.

Ease of Implementation and Lowest Overhead Priority – ***use FCFS*** since the performance of FCFS, RR and FB are very similar while the FCFS is the easiest to implement with the lowest overhead.

Screenshot of Program Compilation and Execution



The program is compiled without any special compiler flags and the output is reported as shown. The output here is from a different run than the output reported in the table, yet the similarity of the performance data is visible.

Normality of the Dataset

A histogram of the dataset generated using the method employed in the program. Each star represents 2000 occurrences of that value in the dataset. As can be seen from the bell-curve like shape of the histogram, the dataset is pretty close to a normally distributed dataset.

```
→ prog4 ./a.out
 1 ********
 2 ***********
 3 ***************
 4 ********************
 5 *************************
 6 *****************************
 7 **********************************
 8 **************************************
 9 ****************************************
10 *****************************************
11 ****************************************
12 **************************************
13 **********************************
14 ******************************
15 *************************
16 ********************
17 ***************
18 ***********
19 ********
20 *****
21 ***
→ prog4
```

A verification of this was done using the raw data and running it through an online normality checker. The software accepts up to 300 datapoints and runs the Kolmogorov-Smirnov test of normality on it. Here is a screenshot of the same. The p-value suggests a good fit with a normal distribution.

The Kolmogorov-Smirnov Test of Normality

Success!

Interpreting the Result

The test statistic (D), which you'll see below, provides a measurement of the divergence of your sample distribution from the normal distribution. The higher the value of D, the less probable it is that your data is normally distributed. The *p*-value quantifies this probability, with a low probability indicating that your sample diverges from a normal distribution to an extent unlikely to arise merely by chance. Put simply, high D, low *p*, is evidence that your data *is not* normally distributed.

It's also worth taking a look at the figures provided for skewness and kurtosis. The nearer both these are to zero, the more likely it is that your distribution is normal.

Your Data

3,8,7,15,3,17,10,
9,7,10,15,1,5,10,
10,16,14,1,12,11
,6,10,17,12,16,4
,13,15,6,5,9,6,1
7,4,4,12,10,8,10
,11,1,15,14,4,13,
10,5,13,8,12,9,9
,11,9,12,7,14,18,
15,19,3,9,11,4,5
,7,7,4,12,14,15,
5,11,11,11,14,20
,17,10,5,9,4,9,1
5,16,7,14,10,11,
13,9,9,8,20,8,9,
10,16,15,16,15,1
1,16,11,8,12,16,
12,4,16,15,13,1
0,7,10,3,10,2,14
,10,4,8,7,1,4,9,1
3,12,14,20,12,9
,13,21,15,3,18,8
,5,8,11,7,14,15,
8,8,9,3,12,10,7,
20,16,17,3,16,1
2,1,10,18,10,4,1
0,10,12,6,9,15,1
7,14,11,9,3,1,10

Distribution Summary

Count : 300

Mean: 10.11667

Median: 10

Standard Deviation: 4.590566

Skewness: -0.040053

Kurtosis: -0.633964

Result: The value of the K-S test statistic (D) is .05901.

The *p*-value is .23763. Your data does *not* differ significantly from that which is normally distributed.