

Process Scheduling Simulator

Jenny Thoppil

Department of Computing and Information Sciences

University of North Florida

Jacksonville, FL-32224-2645

thoj0058@unf.edu

ABSTRACT

The key objective of the Operating System is to optimally manage process scheduling by using scheduling algorithms. Decreasing the user response time and increasing the efficiency of processor achieve optimal system performance. Since each algorithm has its own advantages and disadvantages, it is important to determine the best scheduling policy, given a pool of processes. A Java GUI based process-scheduling simulator was developed in this research. This simulator is a tool to predict the behavior of a set of processes in terms of turnaround and response times. The following scheduling algorithms: First Come First Served (FCFS), Round Robin (RR), Shortest Process Next (SPN), and Shortest Remaining Time (SRT) have been implemented in this research. As an application, instructors for Operating Systems courses at schools could implement this simulator as a visual tool to explain the concept of the scheduling algorithms since visual tools leverage quicker understanding of complex ideas such as CPU scheduling and scheduling algorithms. In industry, this simulator could be used to determine the scheduling policy that would be optimal for any set of jobs with known job arrival and service time. It also acts as a tool to visualize the sequence of jobs before the actual run time.

Keywords

First Come First Served, Round Robin, Shortest Process Next, Shortest Remaining Time

1. INTRODUCTION

A process is defined as a program in execution [2]. It is also described as an entity that can be assigned to and executed by a processor [1]. These definitions emphasize the importance of process scheduling. A careful investigation of process execution order is necessary to optimize the processor utilization and user response time. The most important function of an Operating System is to use process-scheduling algorithms to schedule processes. An Operating System has to manage resources like main memory, I/O devices and processors and schedule the resources among the active processors [1]. A good scheduling algorithm must consider two factors:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
2006 CCEC Symposium, April 12, 2006, Jacksonville, Florida, USA.

Fairness -The processes that are competing for the same resource at the same time must be given approximately equal access to that resource.

Efficiency-The operating system must attempt to maximize throughput and minimize response time.

The scheduling activity of an Operating System can be broken into three functions.

- Long term scheduling is performed when a new process is created to decide whether to add the new process to the set of processes that are currently active.
- Medium term scheduling is used during swapping to decide whether a process should be kept partially in main memory.
- Short term scheduling decides the next process to be executed by the processor. The short term scheduling is defined by system-oriented and user-oriented criteria. For the system-oriented criterion, the effective and efficient utilization of the processor is the most important consideration. The user-oriented criterion relates to the behavior of the system as perceived by the user, that is, response time and turnaround time. Response time is defined as the elapsed time between the submission of a request and the time the response begins to appear as output [1]. Turnaround time is defined as the interval of time between the submission of a process and its completion; it includes the actual execution time plus the time spent waiting for resources, including the processor [1].

2. PROCESS SCHEDULING ALGORITHMS

This research focuses on short-term scheduling algorithms as a way to optimize user response and turnaround time. Short-term scheduling algorithms can be classified as Preemptive and Non-preemptive. In the Non-preemptive algorithm, a process in running state continues to execute until it is terminated, or it blocks itself for I/O. In the preemptive Algorithm, the currently running process may be interrupted and moved to the Ready state by the Operating System [2]. The process scheduling algorithms studied are described below.

2.1 First Come First Served (FCFS)

This is the simplest and non-preemptive scheduling policy. It is also known as the strict queuing system [1]. As each process becomes ready, it joins the ready queue. When the currently running process ceases to execute, the process that has been in

the ready queue the longest is selected for running. Thus FCFS displays extreme fairness to the processes.

There is equal priority given to longer and shorter processes in FCFS. FCFS also may result in inefficient use of both processor and I/O devices. This is because if one process that is processor-bound is followed by many I/O-bound processes, then all the I/O-bound processes have to wait till the processor-bound process finishes execution. When the currently running process leaves the Running state, the ready I/O-bound process will move to the Running state and then get blocked for I/O events. It would be more efficient if the I/O bound process used the I/O device while the processor bound process was running.

FCFS is normally not used by itself. It is usually combined with a priority scheme so that there may be different queues with different priorities, and FCFS is performed on each queue starting with the highest priority queue.

2.2 Round Robin

Round Robin is a preemptive scheduling policy based on a clock. A clock interrupt is generated at regular intervals called time quanta. When an interrupt occurs, the currently running process is placed in the ready queue and the next ready job is selected for execution [1]. This technique is called time slicing because each process is given a slice of time before being preempted.

Short jobs do not suffer in Round Robin. If the time quantum selected is very small, then longer processes will take a very long time to complete and shorter processes will complete execution quickly. If the time quantum selected is very large, then it behaves as FCFS. Hence it is very important to choose the right value of time quantum based on the time required for a typical process function. If the processes in the process queue have low estimated process time then lower time quantum can be chosen, else higher value of time quantum can be chosen.

There is a processing overhead involved in handling the clock interrupt in this technique. The Round Robin does not work well for I/O-bound process as compared to processor-bound ones. Since an I/O-bound process uses a processor for a short period of time and then gets blocked for I/O, it has to join the ready queue after an I/O operation and it does not get to use the entire time quantum. The processor-bound processes, on the other hand, generally use the entire time quantum.

Hence, to avoid the unfairness to I/O-bound processes, the Round Robin method can be modified to have an auxiliary queue along with the ready queue. When an I/O process is blocked, it joins the auxiliary queue instead of joining the ready queue. When a dispatching decision is to be made, processes in the auxiliary queue get preference over those in the main ready queue, thus improving the fairness to I/O-bound processes in the Round Robin technique.

Shortest process next is a non-preemptive scheduling policy in which the process with the shortest expected processing time is selected next

Overall performance is significantly improved in SPN as compared to FCFS and Round Robin. Long processes are not favored as in FCFS.

However, there may be starvation experienced by longer processes if there are many short processes in the pool, thus degrading the response time for the long processes. Also in SPN, an estimate of the required processing time for each process has to be known. If the estimated time is less than the actual running time, then the process may be aborted. If the same jobs run in a production environment, the average of their run times can be used as the expected processing time.

The formula used for expected processing time is as follows [1]

$$S_{n+1} = 1/n \sum_{i=1}^N T_i$$

where T_i = processor execution time for the i th instance of this process

S_i = predicted value for the i th instance.

S_1 = predicted value for first instance; not calculated.

2.3 Shortest Remaining Time (SRT)

The shortest remaining time is the pre-emptive version of SPN. The scheduler selects the process that has the shortest expected remaining processing time from the ready queue.

This scheduler thus works well for short processes and does not favor long processes as FCFS does. It also does not have additional interrupts as in Round Robin, thus reducing overhead. The turnaround time performance of SRT is superior to that of SPN.

In SRT, just as in SPN, the expected processing time of a job has to be known in advance. There is a risk of starvation of longer processes.

2.4 Highest Response Ratio Next

The HRRN technique uses the following ratio in order to select the next process from the ready queue [1] :

$$R = (w+s)/s$$

where R = response ratio

w = time spent waiting for the processor

s = expected service time

It is not possible to know the expected service time ahead of time. But, based on the past history or input from user, the approximate service time can be estimated.

When the current process completes or is blocked, the ready process with the greatest value of R is chosen. This technique accounts for the age of a process.

2.6 Feedback

In this scheduling algorithm, the time spent in execution is taken into consideration. When a process first enters the system, it is placed in the RQ0 queue. After its first pre-emption, it is placed in RQ1 when it returns to the Ready state. Each subsequent time that the process is pre-empted, it is demoted to the next lower-priority queue. The pre-emption times can be varied for each queue. A process scheduled from

RQ0 is allowed to execute for one time unit and then it is preempted; a process scheduled from RQ1 is allowed to execute two time units. Thus there is no starvation of longer processes and turnaround time is lower.

3. PROBLEM DESCRIPTION

In academics, Operating Systems course instructors often use software tools to enhance their teaching. One of the techniques of OS teaching is to modify the kernel code of real operating systems - students are asked to modify some kernel code to see how it changes the system characteristics. This is the most realistic and challenging approach. However, these kinds of projects are often very complicated and require use of dedicated machines [4]. A visual tool that presents complex concepts like multiprogramming and process scheduling helps the students to better understand the dynamics of these concepts. By providing certain inputs, the user should be able to see how they affect the scheduling outcome.

The principle of using scheduling algorithms is not only specific to Operating Systems but also to real-time software projects with jobs that run daily with expected processing times. The simulator can be used to determine the order of the jobs so that the response time is minimized and the performance of the system is increased. Also, the scheduling algorithm that best suits the requirement (for example, fairness to all the jobs or minimum response time) can be determined using the simulator. The simulator makes it easy to experiment with different scheduling strategies. It provides quick feedback about the impact that a particular strategy is likely to have on the progress and completion time of a project. A manager can compare different strategies and choose the strategy which he thinks is best for the next project's setting [6].

4. EXPERIMENTAL SETUP

In this paper, a process-scheduling simulator is implemented using Java Swing. This tool provides options to choose one of the four scheduling strategies: FCFS, Round Robin, Shortest Process Next, and Shortest Remaining Time. The user can input the arrival time and service time for any number of processes and choose a particular scheduling policy. The tool provides the facility to add or delete processes, and gives a visual representation of the process sequence depending on the particular scheduling strategy. It outputs the wait and turnaround time for each process. It also displays the average turnaround time for the scheduling policy. The tool has a feature to find the strategy that provides the least turnaround time, i.e., the best method. Figure 1. is the process scheduling simulator.

Process	Arrival T.	Service	Wait Time	Turnaro...
A	0	3	0	0
B	7	6	0	0
C	4	4	0	0
D	6	5	0	0
E	8	2	0	0

Figure 1: Process Scheduling Simulator

The simulator makes a few assumptions. The resource (processor) is assumed to be available to the processes. The context-switch time between processes is taken as zero for simplicity. Also, the assumption is made that the processes are waiting only for the CPU, not for I/O. Hence I/O time is considered as zero for any process.

SchedulingSimulator.java is the Java GUI and it consists of the following methods:

1. getInfo() – This method gets the data from the table, e.g. arrival time, process name and service time.
2. schedulingSelection() – This method calls the method for the scheduling strategy as chosen by the user.
3. paint() – This method is used to output the graph showing the arrangement of the processes in the Java GUI.
4. FCFS() – This method is used to determine the actual execution time for each process using the First Come First Served scheduling strategy. It determines each process's finish time, response time and turnaround time for this strategy.
5. RoundRobin() — This method is used to determine the actual execution time for each process using the Round Robin scheduling strategy. It determines for each process - the finish time, the response time and the turnaround time for this strategy.
6. SPN()— This method is used to determine the actual execution time for each process using the Shortest Process Next scheduling strategy. It determines for each process - the finish time, the response time and the turnaround time for this strategy.
7. SRT()— This method is used to determine the actual execution time for each process using the Shortest Remaining Time scheduling strategy. It determines for each process- the finish time, the response time, and the turnaround time for this strategy.
8. meanturnaroundtime() – This method invokes each method and finds its turnaround time. It then finds the method that has the least turnaround time and displays that technique's response time and turnaround time.

Using the same methods, another feature is also added where the user can input a file with arrival time and service time and the output is obtained in the form of a file with turnaround time and wait time specified for each process and the mean turnaround time of the method listed in the file.

5. RESULTS

The FCFS technique, shown in Figure 2, reveals that longer process B (Service Time=6) has a normalized turnaround time (TT/Service time =7/6=1.17), which is extremely good. But shorter process E (Service Time=2), which comes after many longer processes, has a high-normalized turnaround time (TT/Service Time=12/2=6). This shows that FCFS favors longer processes. The mean turnaround time is found to be 8.6.

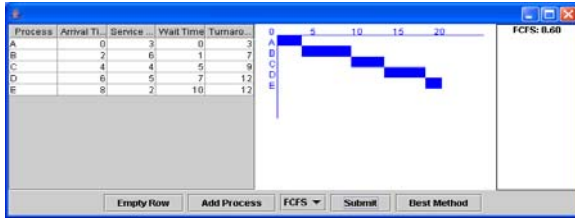


Figure 2: FCFS scheduling

The Round Robin technique with time quantum (TQ= 4) is shown in Figure 3. It shows that longer process B (Service Time=6) has a normalized turnaround time of 2.5 ($TT/Service\ Time=15/6=2.5$). But shorter process E (Service Time=2), which comes after many longer processes, has a normalized turnaround time of 5.5 ($TT/Service\ Time=11/2=5.5$). Round Robin shows lower normalized TT for shorter process than FCFS. Further, when the time quantum is reduced to 1, short process E has a normalized TT = 3.5 which shows that a shorter process is favored. The mean turnaround time is found to be 10.8.

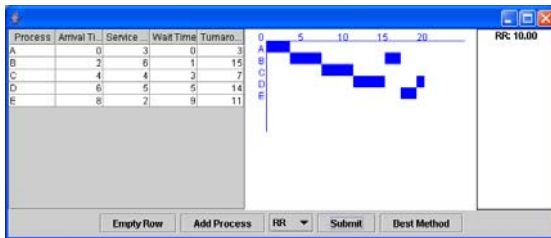


Figure 3: Round Robin scheduling

The Shortest Process Next technique, shown in Figure 4, shows that longer process B (Service Time=6) has a normalized turnaround time of 1.17 ($TT/Service\ Time=7/6=1.17$). Shorter process E (Service Time=2), which comes after many longer processes, has a normalized turnaround time of 1.5 ($TT/Service\ Time=3/2=1.5$). SPN works best for shorter processes; it shows lower normalized TT for shorter processes. It also reveals that if there are many long processes in the ready queue at the same time as the shorter process, the shorter process gets preference over the longer ones. This can be seen for process D whose normalized turnaround time is greater ($TT/Service\ Time=14/5=2.8$). Also, mean turnaround time is found to be 7.6, which is improved compared to FCFS and Round Robin.

The Shortest Remaining Time technique, shown in Figure 5, performs the best among all the methods with the smallest turnaround time of 7.2. It performs the best for shortest process E with normalized turnaround time of 1.00 ($TT/Service\ Time = 2/2 = 1.00$)

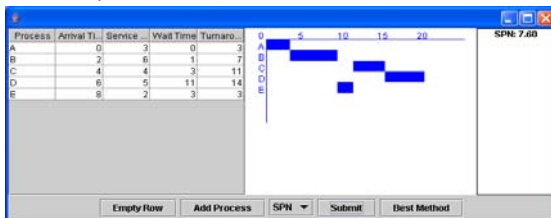


Figure 4: Shortest Process Next scheduling

Many experiments were conducted using only short processes in which case SPN, SRT and FCFS performed the same. When an equal number of long and short processes were used, SRT performed the best in terms of turnaround time.

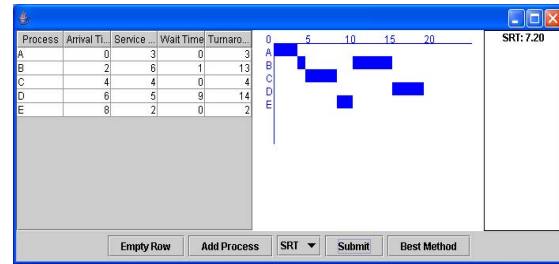


Figure 5: Shortest Remaining Time scheduling

6. CONCLUSIONS

The process-scheduling simulator was used to test FCFS, Round Robin, Shortest Process Next and Shortest Remaining Time scheduling algorithms for a pool of processes and generate a graph of the job execution sequence. The results indicate that the Shortest Process Next algorithm performed the best in most situations. But estimate of the processing time needed to be in known in SPN. SRT performed best when equal number of long and short processes are used, but required more computing time. The simulator was also used to determine the best scheduling algorithm for a particular situation. Instructors for Operating Systems courses can use this tool to explain the concepts of scheduling and scheduling algorithms. This simulator can be used in offices with known job arrival and service time to optimize performance using the best scheduling algorithm. The future goal of this research and experiment is to enhance the tool by combining the FCFS technique, using multiple queues (each with a certain priority), with a priority scheme. A second enhancement will be to use Round Robin in different queues (each with a different time quantum), as used in the Feedback technique, and determine its efficiency.

7. REFERENCES

- [1] A Stallings W., Operating Systems, 4th Edition, Upper Saddle River, NJ Prentice Hall, 2001.
- [2] Tanenbaum, A., Modern Operating Systems, 2nd Edition, Upper Saddle River, NJ Prentice Hall, 2001.
- [3] Tanenbaum A., Woodhull A., Operating Systems Design and Implementation, Upper Saddle River, NJ Prentice Hall, 1997.
- [4] Chan T., A Software Tool in Java for Teaching CPU Scheduling, Methodist College, North Carolina, 2004.
- [5] Robbins S., Robbins K., Empirical Exploration In Undergraduate Operating Systems, University of Texas at San Antonio, 1999.
- [6] Padberg F., A Software Process Scheduling Simulator, Universit/it Karlsruhe, Germany, 2003.