SIMULATION OF CPU SCHEDULING ALGORITHMS

Mr. Umar Saleem and Dr. Muhammad Younus Javed

Computer Engineering Department
College of Electrical and Mechanical Engineering
Peshawar Road, Rawalpindi – 46000
(National University of Sciences and Technology - PAKISTAN)
E-mail : myjaved@yahoo.com
Fax : 051-476190, 474306
Telephone : 051- 478783, 051-561-32966

ABSTRACT

The most important aspect of job scheduling is the ability to create a multi-tasking environment. A single user can not keep either the CPU or the I/O devices busy at all times. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has something to execute. To have several jobs ready to run, the system must keep all of them in memory at the same time for their selection one-by-on. This work involves development of a simulator for CPU scheduling. It has been developed as a comprehensive tool which runs a simulation in real time, and generates useful data to be used for evaluation. A user-friendly and mouse-driven graphical user interface has been integrated. The system has been put through extensive experimentation. The evaluation results extremely useful for the design and development of modern operating systems. This simulator can be used for measuring performance of different scheduling algorithms and for the understanding and training of students.

1. INTRODUCTION

Computer systems supporting multiprogramming or multitasking execute multiple programs/tasks concurrently. One of the objectives of multiprogramming is to maximize resource utilization which is achieved by sharing system resources amongst multiple user and system processes. Efficient resource sharing depends on efficient scheduling of competing processes.

As processor is the most important resource, CPU scheduling becomes very important in achieving the system design goals. Many algorithms have been designed to implement CPU scheduling. Design methods include analytic modeling, deterministic modeling and simulations. Simulation being the most accurate of all is commonly employed for system's performance evaluation despite the fact that it requires complex programming for developing an efficient simulator.

This work involves development of a simulator for CPU scheduling. This simulator can be used for measuring performance of different scheduling algorithms. It simulates: (1) First Come First Serve (FCFS) scheduling, (2) Shortest Job First (SJF) scheduling, (3) priority scheduling, and (4) Round Robin (RR) scheduling.

2. SIMULATOR DESIGN

The system is designed to mimic the dynamic behavior of the system over time. It runs self-driven simulations and uses a synthetic workload that is artificially generated to resemble the expected conditions in the modeled system. This system can also be used to run a trace-driven simulation. Duration of CPU and I/O bursts is read from a data file. The system simulates creation, concurrent execution and termination of processes. It simulates shared utilization of processor, disk drive, printer and magnetic tape drive. It maintains system queues in memory as singly linked lists. These queues include: ready queue, disk drive queue, printer queue and magnetic tape drive queue. It generates useful data that represents the behavior of the system. The data is recorded in a data file for analysis and archival purposes.

A user-friendly and mouse-driven Graphical User Interface (GUI) has been integrated. It provides the user an opportunity to opt either for a pictorial view of concurrent execution of processes or for a real time Gantt chart representation of resource utilization. The GUI displays real time view of system queues and timings of CPU and I/O bursts. A selection can be made from a menu displaying four options for the different CPU scheduling algorithms. Simulation can be run either in continuous mode or in step mode. Continuos mode is the default mode. Step mode provides the opportunity to run simulation in steps. The processing of a batch can be paused temporarily without corrupting the output data. These features make the system attractive for experimental as well as academic use.

FCFS is the simplest scheduling strategy discipline. The workload is simply processed in the order of arrival. Implementation of the FCFS scheduler is quite straightforward. It can be easily managed with a First In First Out (FIFO) queue. When a process enters the ready queue, its Process Control Block (PCB) is linked onto the tail of the ready queue. When CPU is free, it is allocated to the process at the head of the ready queue. However, it dose not take into consideration the state of the system and the resource requirements of the individual scheduling entities, because of it this scheduler may result in poor performance, lower throughput and longer average waiting times. Short jobs may suffer considerable turnaround delays and waiting

0-7803-6355-8/00/ $10.00 © 2000 IEEE

times when one or more long jobs are in the system [1,2]. SJF is a scheduling discipline in which the next scheduling entity, a process, is selected on the basis of the shortest execution time of its next CPU burst. Whenever the SJF scheduler is invoked, it searches the ready queue to find the process with the shortest next execution time. SJF scheduler is invoked whenever a process completes execution or the running process surrenders control to the operating system [1,2].

The RR scheduling algorithm, which is also known as time slice scheduling, is designed especially for time sharing systems. The CPU scheduler goes around the ready queue allocating the CPU to each process for a time interval of up to one time slice or time quantum. The ready queue which is treated as a circular queue is kept as First In First Out (FIFO) queue of processes. New processes are added at the tail of the ready queue and CPU scheduler picks the process from the head of the queue. After expiry of the time quantum the process is preempted and placed at the tail of the ready queue [1,2,3]. In real systems, however, the context switching would take place on expiry of each time slice. RR scheduling achieves equitable sharing of system resources. Short processes may be executed within a single time quantum and thus exhibit good response time. Long processes may require several quantum and thus be forced to cycle through the ready queue for a longer time before completion [1].

In priority scheduling, each process in the system is assigned a priority level and the scheduler chooses the highest priority process from the ready queue. Equal priority processes are scheduled in FCFS order. In this case there is a possibility that low-priority processes be effectively locked out by the higher priority ones. In other words completion of a process within finite time of its creation cannot be guaranteed with this scheduling policy. In this scheme, however, the typical performance criteria are traded with the priority objectives. In other words the priority [1] criteria determines the performance criteria. The order of completion of jobs is determined by the way the jobs are selected by the scheduler.

3. SOFTWARE DEVELOPMENT

The Simulator CPU Scheduling Algorithm (SCSA) has been developed with a view to develop a software tool which can be used for the study and evaluation of CPU scheduling algorithms. SCSA has been developed as a comprehensive software package which runs a simulation in real time, generates useful data to be used for evaluation and provides a user-friendly environment. Software design strategy is functional oriented and design is modular in nature. The system is designed to run on PC under MSDOS or WINDOWS environment. It provides the user an opportunity to simulate a multiprogramming and multi-user environment on a PC. Initialization and set up module of SCSA performs set up operations. It reads simulation data from a user-defined data file. In accordance with the data it makes Process Control Blocks (PCB) and develops ready queue in memory. On user instruction the control is then handed over to relevant simulation module.

3.1 SIMULATION MODULES

Four simulation modules have been utilized in the design and development of SCSA: (1) (w) FCFS module selects the process from head of the ready using FIFO mechanism queue and simulates its execution in time. The process then either terminates or is placed in a device queue; (2) (b) In SJF scheduling module, PCB are linked in the ready queue in the order of next CPU bursts. With the PCB having shortest burst placed at the head of the ready queue. Processes are selected from the head of the ready queue and its real time execution is simulated. After the execution, the process either terminates or requests for Input/output (I/O) operation. Next process with the shortest CPU burst is then selected for execution; (3) (c) Ready queue is maintained as a FIFO queue in RR scheduling module. Processes are selected from the head of the ready queue. Processes being executed are preempted on expiry of time quantum which is a user-defined data. A preempted process is linked at the tail of the ready queue. Process completing CPU burst before the expiry of time quanta either terminates or is placed in a device queue and (4) (d) Priority scheduling module links PCBs in the ready queue in the order of a user-defined priority criteria with the process having highest priority placed at the head. Processes are selected from the head of the ready queue and execution is simulated. On completion of CPU burst the process either terminates or requests for an I/O operation. Simulation performance data is recorded during execution in the appropriate file for every module.

3.2 I/O AND GUI MODULE

SCSA requires a user to provide it information regarding number of processes in a batch, the sequence and timing of CPU and I/O bursts in each process. During set up it reads a user-defined input data file from current directory. The user is required to develop this file as a text file and enter data of each process in a specified format. SCSA records algorithm performance data in an output text file during execution of simulations. The data provides performance information such as: (1) execution times of CPU and I/O bursts of each process, (2) waiting times of each process, (3) turnaround times of each process, (4) average waiting time of batch, (5) CPU utilization by each process, (6) CPU utilization by the batch, and (7) total execution time of batch. GUI module provides a menu driven interactive environment. First menu, which is called main menu, shows four buttons and asks the user to select an algorithm for simulation by clicking the appropriate button. In case the user selects RR scheduling algorithm option, SCSA prompts the user by displaying a pop up window to enter value of time quantum. It also provides a button to select a screen which displays performance results of a batch. On selection of an algorithm, SCSA displays a new screen. This presents a simulation window, ready queue and device queue windows, concurrent processing data windows, a timer window and an assortment of control buttons. Simulation window has three display modes: (1) a display mode which

provides a pictorial view of activities of CPU and I/O devices, (2) a chart mode which paints a real time Gantt chart of concurrent processing activity, and (3) a no-display mode. Real time queues are displayed in ready queue and device queue windows. Processing windows display the system state and CPU/burst timings. During processing of a batch elapsed time is displayed in timer window. Control on processing of a batch can be exercised through a number of control buttons which are: (1) display mode buttons, these can be clicked with the mouse to select display mode option, (2) main menu button which can be used to terminate processing of a batch and to go back to main menu, (3) a start button to commence processing, (4) a pause button, on click, it pauses the processing till the time it is clicked again, and (5) two buttons to toggle between continuos mode and step mode.

4. EVALUATION RESULTS AND DISCUSSION

The simulator has been used to carry out extensive experimentation. Simulations of algorithms generated useful data that has been used to draw results. Evaluation performance of each CPU algorithm is discussed in the succeeding paragraphs.

4.1 FIRST COME FIRST SERVE (FCFS) SCHEDULING

Results obtained by simulation of FCFS algorithm has confirmed that the execution of FCFS scheduling produce smaller computational overheads. It gives poor performance, lower throughput and longer average waiting times. A batch of 50 processes is shown in Figure 1(a). CPU burst lengths of the batch varies from 0.1 to 2.0 sec. The jobs are submitted to the processor in the order of their arrival to the system and are executed and completed by the processor in the same order. In this scheduling scheme the criteria for the selection and scheduling of the jobs to the processor is the order of their arrival into the system, hence this scheme does not take into account the time required for completion of individual processes. Because of this, short jobs if submitted after some longer jobs have to wait for longer times.

The relationship between average CPU burst length and average waiting time is non-linear. Figure 1(b) shows average CPU burst lengths of 10 batches plotted against average waiting times. These results show that the average waiting time increases non- linearly with the increase in average burst lengths.

If a few relatively long jobs are present in the system shorter jobs experience longer waiting time. Figure 1(c & e) shows two batches of 50 processes each. In case of batch 1(Figure 1(c, d)), there is a mix of short and long jobs. The average CPU burst length is 1.25 sec with a max variance of 0.25 sec. The resulting waiting times range from 0 to 60 sec, with an average of 30 sec. Batch 2 (Figure 1(e, f)) is designed by replacing three processes of batch 1 with processes having CPU burst lengths of 8, 10 and 11 sec respectively. These long jobs, as submitted to the system earlier in the batch, badly affect the completion of short jobs. As their execution is pushed almost 29 sec behind, resulting in a delay in completion of 29 sec. In this case waiting times range from 0 to 90 sec, at an average of 53 sec (an increase of almost 56.6% as compared to batch .

4.2 SHORTEST JOB FIRST (SJF) SCHEDULING

A batch of 50 processes is shown in Figure 2(a). In this batch CPU burst lengths vary linearly from 0.1 to 2 sec. The jobs are submitted to the system in the order of their arrival and are executed and completed in the order of their CPU burst lengths with a priority for the shorter job first as shown in Figure 2(b).

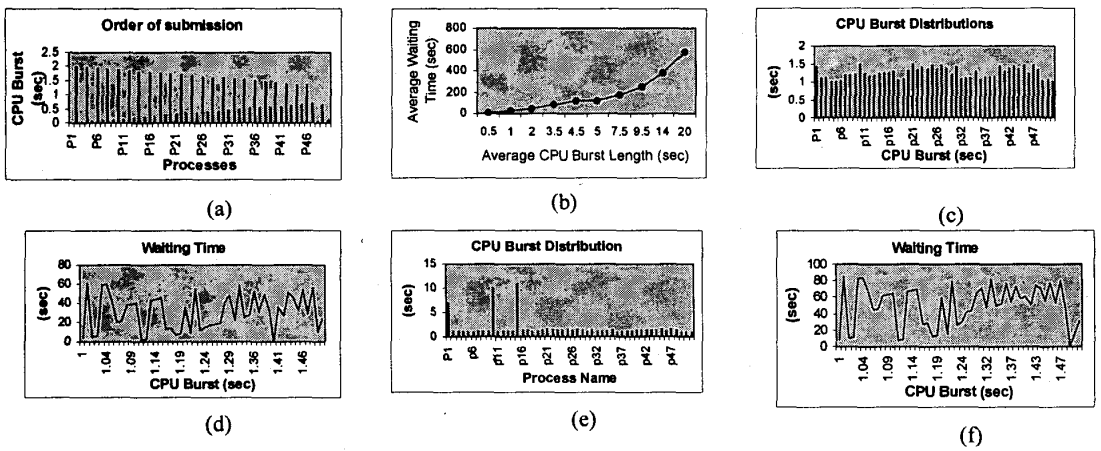
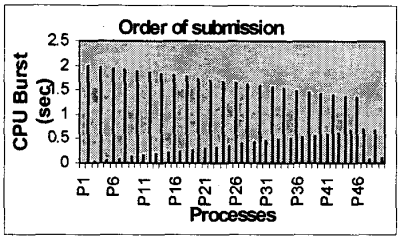
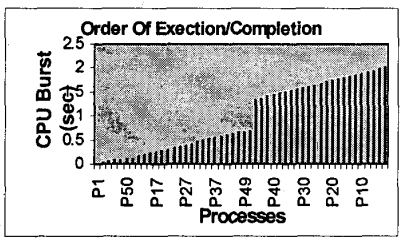


Figure 1 FCFS Results

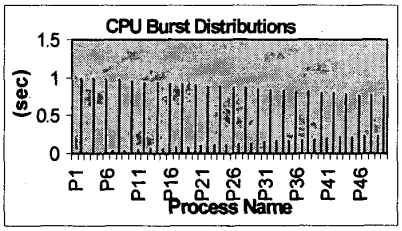
SJF scheduling is an optimal scheduling discipline in terms of minimizing the average waiting time of a given workload. However, preferred treatment of short processes in SJF scheduling tends to result in increased waiting times for long processes. Thus, if there is a continuos inflow of short processes in the system there is a possibility that long processes may get stranded in the ready queue forever. In Figure 2(c) a batch of 50 processes is shown having CPU burst lengths ranging from 0.1 to 1 sec with an average of 0.5 sec. In case the batch is processed with a FCFS algorithm the resulting waiting/turnaround times experienced by the individual processes will be in accordance with their relative positions in the batch. Whereas if the batch is processed with a SJF algorithm the resulting waiting/turnaround times will be as such that short jobs will be executed earlier than long jobs.



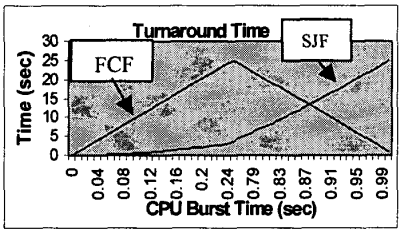
(a)



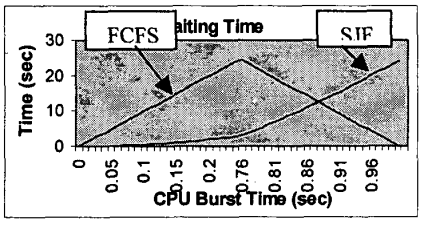
(b)



(c)



(d)



(e)

Figure 2 SJF Results

Figure 2(d) and 2(e) show that long jobs in this case have experienced discrimination and have been forced to stay longer in the system. Short jobs experience better waiting times with SJF scheduling in comparison with FCFS scheduling. Medium burst length processes experience almost equal waiting times in both the cases. Whereas long processes result in better waiting time with FCFS scheduling when compared to SJF scheduling. However, average waiting time of the batch has improved in comparison with the FCFS scheduling. Figure 3(a) shows a batch of 50 processes with CPU burst lengths ranging from 1 to 15 sec with an average burst length of approximately 6 sec. Execution of this batch with FCFS and SJF scheduling shows same results as shown in Figure 2(d) and 2(e). Relationship between average waiting times experienced by FCFS and SJF scheduling is nonlinear. It has been observed that average waiting times are similar in both the cases when all the processes in a batch have similar CPU burst lengths. But as the burst lengths in a batch becomes increasingly different, which is indicated by growing variance in the figure, the relationship gradually becomes nonlinear with a definite improvement in the waiting times of SJF scheduling.

4.3 ROUND ROBIN (RR)

Figure 3(a) shows a batch of 50 processes having CPU burst lengths ranging from 0.1 to 2.0 sec, with an average of 1 sec. This batch is processed with a time quantum of 0.5 sec. As is shown in the Figure 3(b), processes having burst lengths less than the time quantum are executed completely in one slice, whereas processes having burst lengths longer than time quantum are sliced down by the algorithm in burst lengths of 0.5 sec each. A single process rotates many times between processor and ready queue before completion. Long processes thus are forced to stay longer in the system and are subjected to longer waiting times. The completion of the batch took a little longer than it would have taken in the case of FSCF scheduling because of scheduling overhead. On the other hand, all processes short or long get equitable response time and share of the processor time making this algorithm attractive for time sharing systems. Figure 3(c) shows CPU burst distributions. Shorter processes complete before the expiry of time quanta, so such processes if present in the batch are completed earlier. Medium burst length processes consume

a few time slices before completion. Whereas long processes take greater number of time slices for completion resulting in longer completion time. The algorithm appears to exhibit the selective properties of SJF scheduling but the equitable response time mark the difference. Figure 3(c & d) shows yet another batch that consists of 52 processes, the burst length range from 0.78 to 12 sec. Batch is processed with a time quantum of 0.5 sec. Order of completion is shown in Figure 3(d).

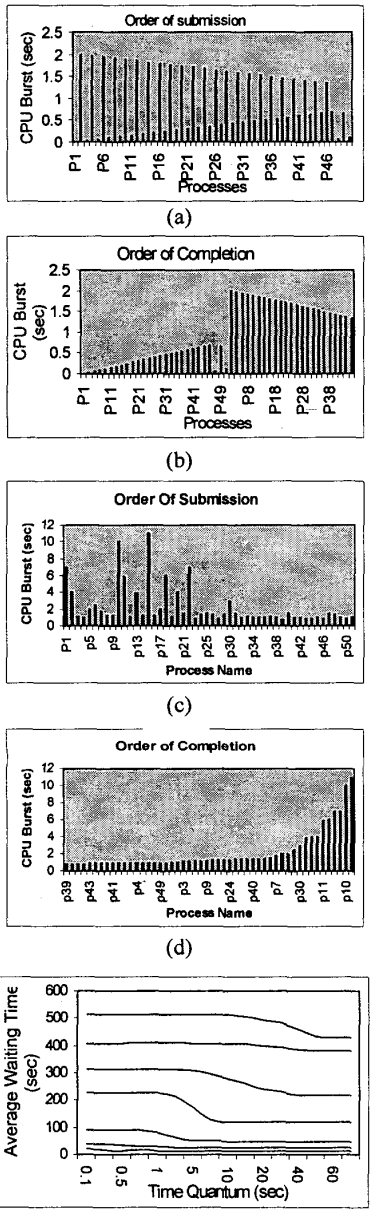


Figure 3 RR Results

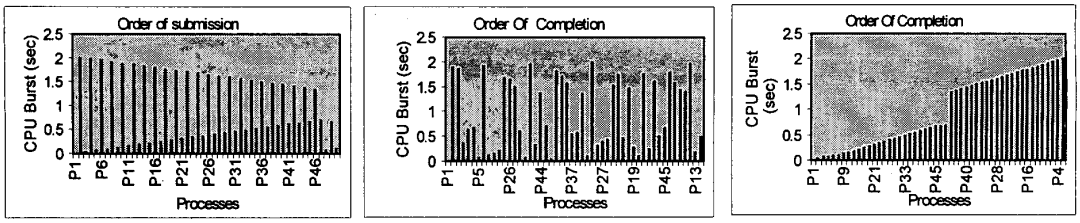
The relationship between the time slice and performance is nonlinear. Too short a time slice may result in significant overhead due to the frequent timer interrupts and context switches. A 2 ms time slice in a system where a context switch takes 500 μs, amounts to a 25% overhead. On the other hand, too long a time slice reduces the preemption overhead but increases response time. For example, a 100 ms time slice in a system with 50 active users implies 5 seconds response time. In the extreme, a very long time quantum transforms a RR scheduler into a FCFS scheduler, thus further deteriorating throughput and average waiting times [1]. Figure 3(e) shows 7 batches of processes, each with a different average CPU burst length. The batches were executed repeatedly with different time quantum (i.e. 0.1, 0.2, 0.3, 0.4, 0.5, 0.75, 1, 2.5, 5, 7.5, 10, 15, 20, 30, 40, 50 and 60 sec). The response in terms of average waiting times are plotted. It is evident that a small time quantum results in higher average waiting times, which is caused by greater degree of context switching overheads. However, too small a time quantum in relation to the burst lengths results in the so-called processor sharing, which gives better response time. With the increase in time quantum switching overhead decreases, resulting in improved average waiting times. However, it increases the response time. Then further increases in the time quantum make the time slice longer than the longest burst in the batch, at this point the scheduler degenerates into FCFS scheduling.

4.4 PRIORITY SCHEDULING

Figure 4 shows a batch of 50 processes with CPU burst lengths ranging from 0.1 to 2 sec. The batch is processed with three different priority schemes. Figure 4 (a) depicts a case where the jobs are prioritized in accordance with their operational significance. Whereas Figure 4 (b) shows a case where the priority criteria is the execution in the order of arrival of the jobs into the system. With this criterion, priority scheduling behaves like FCFS scheduling. Figure 4(c) shows a case where the jobs with shorter CPU burst lengths are assigned higher priorities. The criterion makes the priority scheduling behaves like SJF scheduling.

5. CONCLUSION

CPU scheduling algorithms maximise utilization of the CPU and minimise resource idle time. In order to create a multiprogramming environment, SCSA (Simulator CPU Scheduling Algorithm) was developed which contained a number of simulation modules. Four CPU scheduling strategies were integrated with the SCSA for obtaining real time performance results. Each module executes the selected algorithm and displays results in the form of Gantt charts using an effective graphical user interface. The system has been put through extensive experimentation. Results have shown that the execution of FCFS scheduling produce smaller computational overheads because of its simplicity, but it gives poor performance, lower throughput and longer average waiting times. SJF is an optimal scheduling discipline in terms of minimizing the average waiting time of a given workload. However, preferred treatment of short processes in SJF scheduling tends to result in increased waiting times for long processes in



(a) (b) (c)

Figure 4 Priority Scheduling Results

Comparison with FCFS scheduling. Thus, there is a possibility that long processes may get stranded in the ready queue because of continuous arrival of shorter processes in the queue. RR scheduling achieves equitable sharing of CPU. Short processes execute within a single time quantum and thus exhibit good response time. It subjects long processes to relatively longer turnaround and waiting times. In case of priority-based scheduling there is a possibility that low-priority processes be effectively locked out by the higher priority ones. In other words completion of a process within finite time of its creation cannot be guaranteed with this scheduling policy. Experimental results are extremely useful for the design and development of modern operating systems.

6. REFERENCES

[1] Milenkovic, M., *Operating System Concepts and Design*, McGraw Hill, International Edition, 1992, pp. 65-76.

[2] Silberschatz, A., Peterson, J. L., and Galvin, P.B., *Operating System Concepts*, Addison Wesley, 3rd Edition, 1991, pp. 108-118.

[3] Beck, L. L., *System Software*, Addison Wesley, 2nd Edition, 1990, pp. 322-324.