Performance Evaluation of CPU Scheduling by Using Hybrid Approach

Jyotirmay Patel

Research Scholar, Bhagwant University, Rajasthan, India.

A.K.Solanki

Professor, Department of Computer Sc. & Engg, Bundelkhand University, India.

Abstract

Central Processing Unit (CPU) scheduling plays crucial role by switching the CPU among various processes. The problem of scheduling which computer process run at what time on the central processing unit (CPU) or the processor is explored. Some CPU scheduling algorithms has been elaborated and assessed on the basic CPU scheduling objectives i.e; average waiting time, turnaround time etc. These will form the base parameters in making a decision for the suitability of the given algorithm for a given objective. Many algorithms have been developed for the CPU scheduling of a modern multiprogramming operating system. Our research work involves the design and development of new CPU scheduling algorithm (the Hybrid Scheduling Algorithm). This work involves a software tool which produces a comprehensive simulation of a number of CPU scheduling algorithms. The tool's results are in the form of scheduling performance metrics.

Index terms: CPU scheduling, turnaround time, Hybrid algorithms, waiting time.

1.0 Introduction

Scheduling is a fundamental function of an operating system. The main concept is to share computer resources among a number of processes. Almost each computer resource is scheduled before use. The CPU is one of the primary computer resources, so its scheduling is essential to an operating system's design. CPU scheduling decides which processes execute when there are multiple run-able processes. CPU scheduling is important because it plays an important role in effective resource utilization and the overall performance of the system. Scheduling is a branch of the topic of Operational Research. In terms of scheduling, we are considering how best to schedule multiple jobs for processing by a single machine (CPU). It is a relatively easy problem as compared to the problem of scheduling multiple jobs for multiple machines. Some idea of the difficulties associated with this latter problem can be garnered by studying 'job shop' and 'flow shop' scheduling problems.

2.0 Related Work

There are many existing CPU scheduling algorithm simulators. Some are more user-friendly than others. Some are command-line driven whilst others have a GUI (Graphical User Interface). Let us briefly mention some of the simulators that are available.

Suranauwarat developed a simulator which produces a simulation of various scheduling algorithms for a single CPU. A user can run this simulator with predefined scheduling parameters and can also customize parameters for a set of processes. The simulator works in two operating modes. The first mode is 'simulation mode' and the second one is called 'practice mode'. In simulation mode, the user can interact with the simulation during process execution. A user can start and stop the simulation whenever he or she likes. A user can also monitor the simulation straight through from the beginning until the end. By using the simulator in simulation mode, the user could achieve a better conceptual understanding of the CPU scheduling algorithms. In practice mode, the user can predict when and for how long each process is in a particular state. The user is also able to predict why a process is in that state through a very good graphical user interface. The user is also provided with the facility to check whether his or her answer is correct or not at any time during practice. One drawback of this research work is that it is only limited to traditional scheduling policies.

Another drawback is that it does not provide any comparative results of different scheduling policies. A *scheduling simulator named CSCI 152 CPU Scheduling Algorithm Simulator*. This simulator is a server-side program that allows the user to interact with it via its web form. It provides a very good graphical web-based interface. It gives a comparison of the performance of three scheduling algorithms (the FCFS, RR, and SJF scheduling algorithms) for the same set of processes. The limitations of this simulator are that it only works for three scheduling algorithms and that the comparative analysis only relates to response times. *The Tran's Scheduling Algorithm Simulator* supports a number of scheduling algorithms such as FCFS, RR, SJF, SRTF and HRR. The time quantum is program coded and taken as 1 for each set of processes. It lets the user

create a personal set of processes. However, this simulator uses a simple process model, that is, there is only one CPU burst per process. The drawback of this simulator is that the programmed time quantum is one and this causes a number of context switches. This simulator does not produce a comparative performance analysis of the different CPU. Each of the above simulators uses a Gantt chart to animate which process is using the CPU at what time. This approach is fine when the process model is one CPU burst per process.

The author decided to develop their own scheduling simulator. A new program was necessary as no system was available to offer the desired functionality. The resultant program needed to be an efficient, pictorial and user friendly simulation to depict a multiprogramming environment on a PC-based platform. There existed a strong need for such a system which would enable a PC-based user to analyze a multiprogramming operating system environment, for the purpose of either analyzing the design of a CPU scheduling system or studying the science of process scheduling. The system described here has been designed with a view to fill the gap, offering a modest simulation of a PC-based multiprogramming environment. The simulator is unique in a number of respects. It should be emphasized that there is a valuable 'spinoff' to this part of our work in that the simulator can be used in the classroom.

3. 0 Scheduling Objectives

Many objectives must be considered in the design of a scheduling discipline. In particular, a scheduler should consider fairness, efficiency, response time, turnaround time, throughput, etc.

3.1 Fairness

Fairness is important under all circumstances. A scheduler makes sure that each process gets its fair share of the CPU and no process can suffer indefinite postponement. Note that giving equivalent or equal time is not fair. Think of *safety control* and *payroll* at a nuclear plant.

3.2 Policy Enforcement

The scheduler has to make sure that system's policy is enforced. For example, if the local policy is safety then the *safety control processes* must be able to run whenever they want to, even if it means delay in *payroll processes*.

3.3 Efficiency

Scheduler should keep the system (or in particular CPU) busy cent percent of the time when possible. If the CPU and all the Input/Output devices can be kept running all the time, more work gets done per second than if some components are idle.

3.4 Response Time

A scheduler should minimize the response time for interactive user.

3.5 Turnaround Time

A scheduler should minimize the time batch users must wait for an output.

3.6 Throughput

A scheduler should maximize the number of jobs processed per unit time. A little thought will show that some of these goals are contradictory. It can be shown that any scheduling algorithm that favours some class of jobs hurts another class of jobs. The amount of CPU time available is finite, after all.

4.0 Brief Overview of Scheduling Algorithms

4.1 First-Come, First-Served (FCFS): Processes are assigned the CPU in the order they request it.

4.2 Round-Robin (RR): Each process is given a limited amount of CPU time, called a time slice, to execute. If the required CPU burst of the process is less than or equal to the time slice, it releases the CPU voluntarily. Otherwise, the scheduler will preempt the running process after one time slice and put it at the back of the ready queue, then dispatch another process from the ready queue.

4.3 Shortest-Job-First (SJF) Non-preemptive: When the CPU is available, it is allocated to the process that has the smallest next CPU burst. *SJF Preemptive:* When the CPU is available, it is allocated to the process that has the shortest remaining CPU burst. When a process arrives at the ready queue, it may have a shorter remaining CPU burst than the currently running process. Accordingly, the scheduler will preempt the currently running process.

4.4 Multilevel Feedback Queues (MLFQ):

There are several ready queues, each with different priority. When the CPU is available, the scheduler selects a process from the highest-

priority, non-empty ready queue. Within a queue, it uses RR scheduling. The scheduler adjusts the priority of a process dynamically, for example, to reflect resource requirements (e.g., being blocked awaiting an event) and the amount of resources consumed by the process (e.g., CPU time). Processes are moved between ready queues based on changes in their priority. When a process other than the currently running process attains a higher priority, the scheduler will preempt the currently running process and add it to the appropriate ready queue.

4.5 Priority Scheduling (PS)

The PS algorithm associates with each process a priority and the CPU is allocated to the process based on their priorities. Usually, lower numbers are used to represent higher priorities. The process with the highest priority is allocated first. If there are multiple processes with same priority, typically the FCFS is used to break tie.

4.6 Highest Response Ratio Next scheduling algorithm (HRRN)

Proposed by Brinch Hansen is a to avoid limitations of SJF algorithm. It is similar to Shortest Job Next (SJN) in which the priority of each job is dependent on its estimated run time, and also the amount of time it has spent waiting in the ready queue.

5.0 Research Requirements

5.1 Hardware Requirements

- PC with PENTIUM II & above
- 1 GB RAM and above
- 120 GB HARD DISK
- STORING DEVICES

5.2 Software Requirements:

- WINDOWS XP and above
- .NET FRAMEWORK

5.2 Platform Used

Choice of platform

The selection of software involves two major decisions namely

a) Selection of Front - end

b) Selection of Back- end

C# language is the appropriate choice for the Front -end. For the Back - end, we have used flat file since this is readily available in all stations.

6.0 Methodology Used

6.1 Algorithm Evaluation

How do we select a CPU-scheduling algorithm for a particular system? There are many scheduling algorithms, each with its own merits and demerits based on different parameters. As a result, selecting algorithms can be difficult. The first problem is defining the criteria to be used in selecting an algorithm. Criteria are often defined in terms of CPU utilization, response time or throughput. To select an algorithm, we must first define the relative importance of these measures. Our criteria may include several measures, such as:

- Maximize throughput such that turnaround is (on average) linearly proportional to total execution time.

Once the selection criteria have been defined, we are then going to evaluate the various algorithms under consideration.

Basically following are the different evaluation methods which are commonly used:

- Deterministic Modelling Evaluation Method
- Queuing Models for Evaluation
- Simulation for evaluation

6.2 Simulation for evaluation

In our work simulation is being used. This is used to get a more accurate evaluation of scheduling algorithms. Simulation involve programming a model of the computer system. Software data structures represent the major components of the system. The simulator has a variable representing a clock ; as this variable's value is increased, the simulator modifies the system state to reflect the activities of the devices, the processes and the scheduler. As the simulation executers, statistics that indicate algorithm performance are gathered and printed. The advantages of simulation are:

- It produces accurate results for its inputs.
- One of the primary advantages of simulators is that they are able to provide user with practical feedback when designing real world systems. This allows the designer to determine the correctness and efficiency of a design before the system is actually constructed. Consequently, the user may explore the merits of alternative designs without actually physically building the systems. By investigating the effects of specific design

decisions during the design phase rather than the construction phase, the overall cost of building the system diminishes significantly.

Disadvantages

The design, coding and debugging of the simulator can be a major task.

6.3 Implementation Method

Even a simulator is of limited accuracy. The only completely accurate way to evaluate a scheduling algorithm is to code it, put it in the operating system, and see how it works. This approach puts the actual algorithm in the real system for evaluation under real operating conditions.

Limitations

This approach is very expensive. The expense is incurred non only in coding the algorithm and modifying the operating system to support it as well as its required data structure, but also in the reaction of the users to a constantly changing operating system.

7.0 Proposed Scheduling Algorithm by Using Hybrid Performance Parameters

Highest Response Round Ratio Next (HRRRN) = Highest Response Ratio Next (HRNN) + Round Robin (RR)

Highest Response Ratio Next **s**cheduling algorithm proposed by Brinch Hansen is a to avoid limitations of SJF algorithm. It is similar to Shortest Job Next (SJN) in which the priority of each job is dependent on its estimated run time, and also the amount of time it has spent waiting in the ready queue. Jobs which gain higher priority the longer they wait, which prevents process starvation. In fact, the jobs that have spent a long time in waiting, can compete against those jobs estimated to have short run time.

Response Time= (Waiting Time + Run Time) /(Run Time).

- We calculate response ratio for all the processes which are not executed completely.
- The processes which have the highest response ratio will be executed next.
- The processes are executed in ROUND ROBIN manner.
- In our algorithm the quantum for our algorithm is defined by system automatically which is calculated by using the formula.

  Quantum = (average burst time / 1.5)
- Thus processes are executed in ROUND ROBIN manner but the next process to execute is of course with highest response ratio.
- In this way the process which have lower burst time but already waited for long time will get the chance to execute in the next opportunity.

8.0 Market Potential and Competitive Advantages

Over time, the computational strength available to users has rapidly increased through the development of faster individual CPUs. As this has occurred, there has continued to be a development of computationally intensive applications that still yearn for further extensions of computational power to do their work more quickly. Faster and inexpensive networking speeds, expansion of computer sales, and rapid growth in popularity of the Internet indicate that, in theory, there exists an additional supply of reachable computing power for such applications. It is therefore a possibility that extra computational capabilities could come from the collaboration of networked computers willing to donate their resources, particularly their CPU cycles. Under this scenario, there would be users wishing to run their applications and also hosts with extra CPU cycles to give away. Once these respective sides are known, there needs to be a means of matching members of these two parties. One decision that needs to be made is whether or not to assign a CPU to a particular task. This is typically defined as a problem of resource allocation.

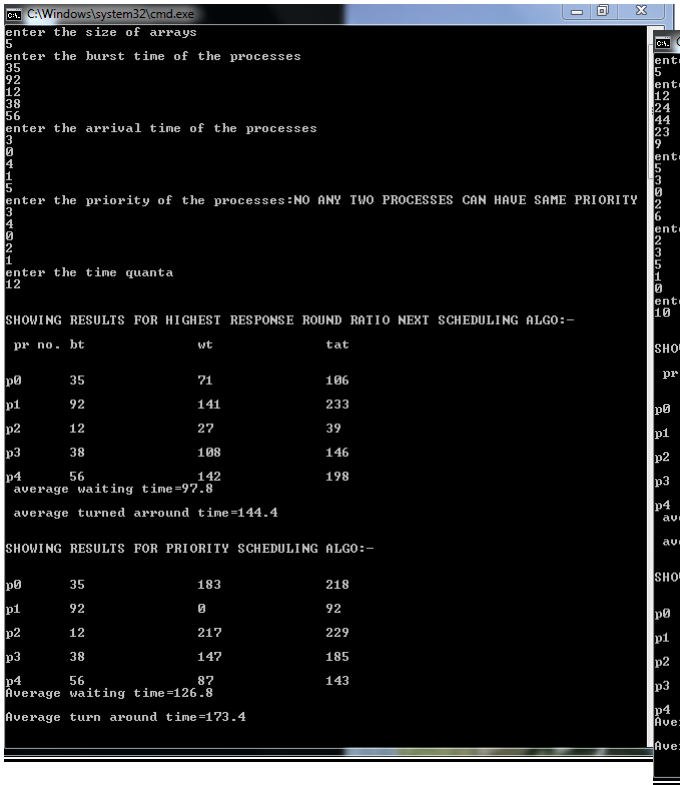
However, there is also often a need to know when exactly a computation should be performed. When this time element is a factor in addition to the allocation of the resource, this is termed to be a scheduling problem. As such, scheduling is just a more specific instance of resource allocation. Scheduling in distributed computer systems has long been researched, both in single and multiple CPU situations. Many proposed solutions have revolved around the use of heuristics, and others involve the use of deterministic mathematical models. Particularly, research in one branch has involved the use of economic algorithms in what have been called market-based scheduling strategies.

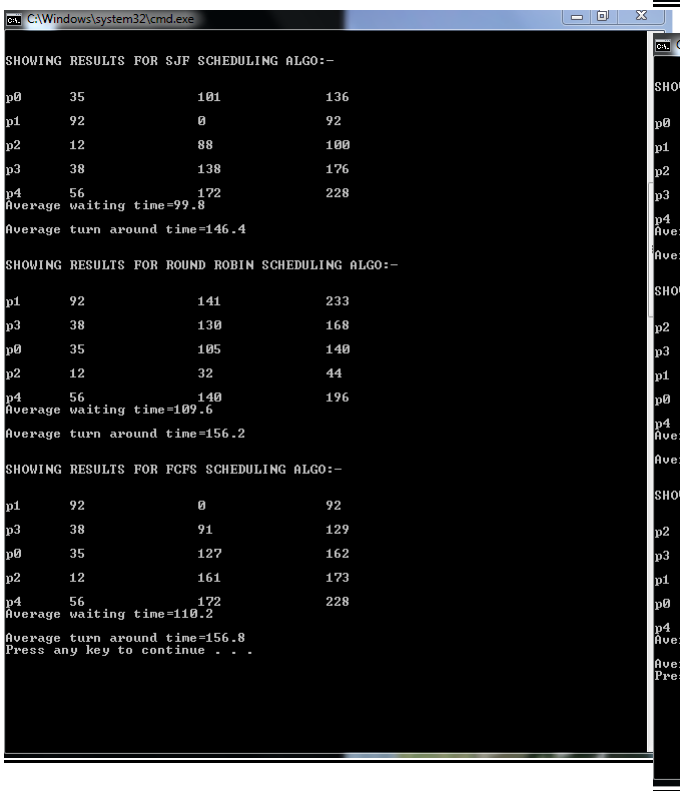
9.0 Screenshot/Output of the Proposed Algorithm

Let us consider three examples/case:

Example1:-

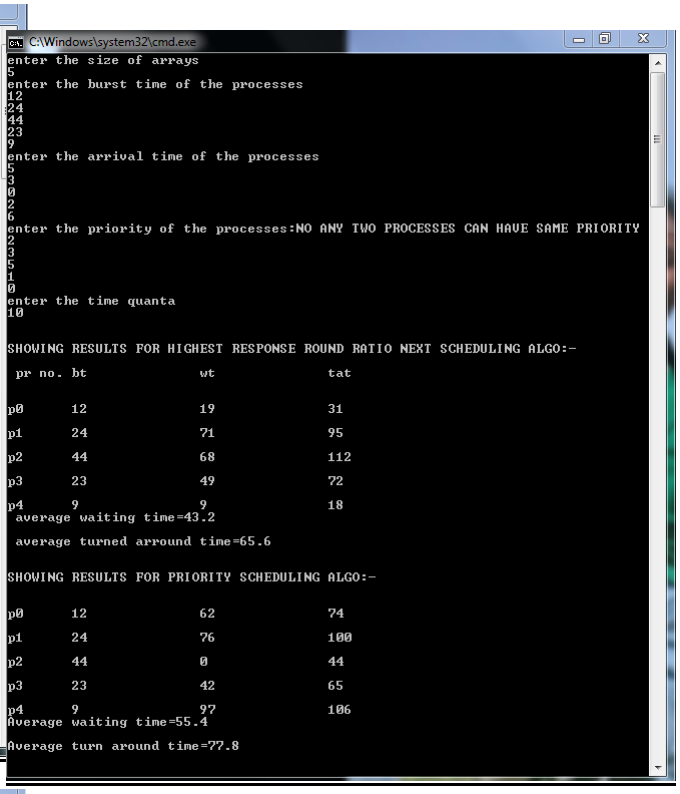
Screenshot 1& 2

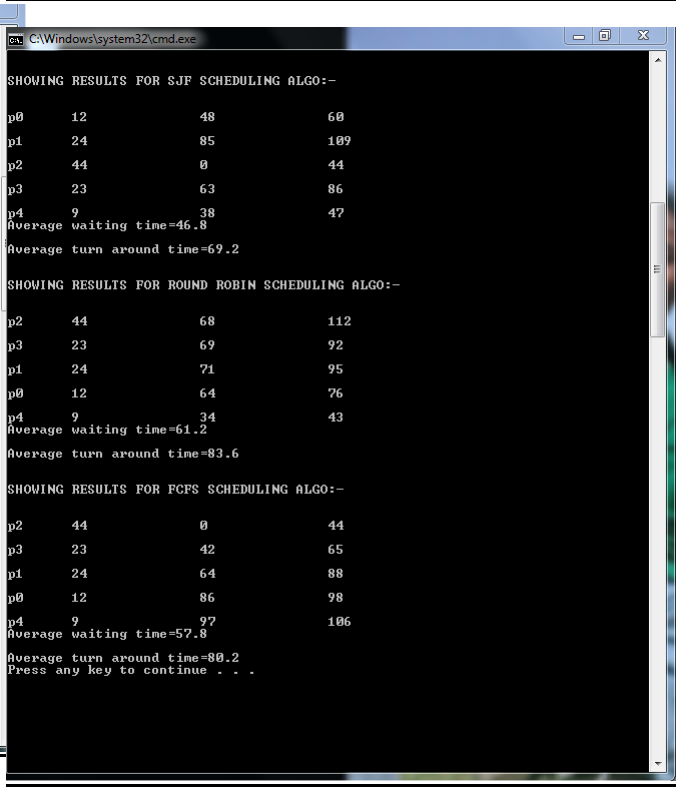




Example2:-

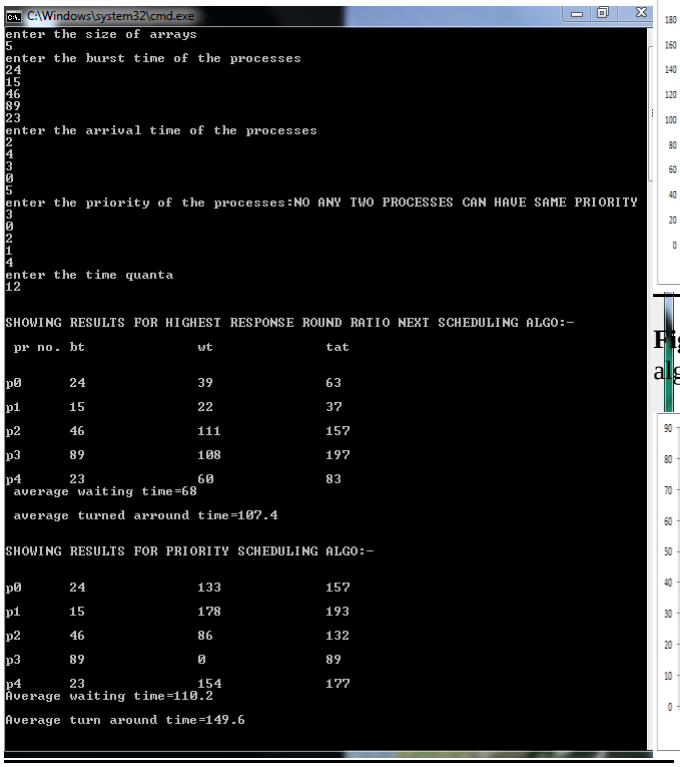
Screenshot 3& 4

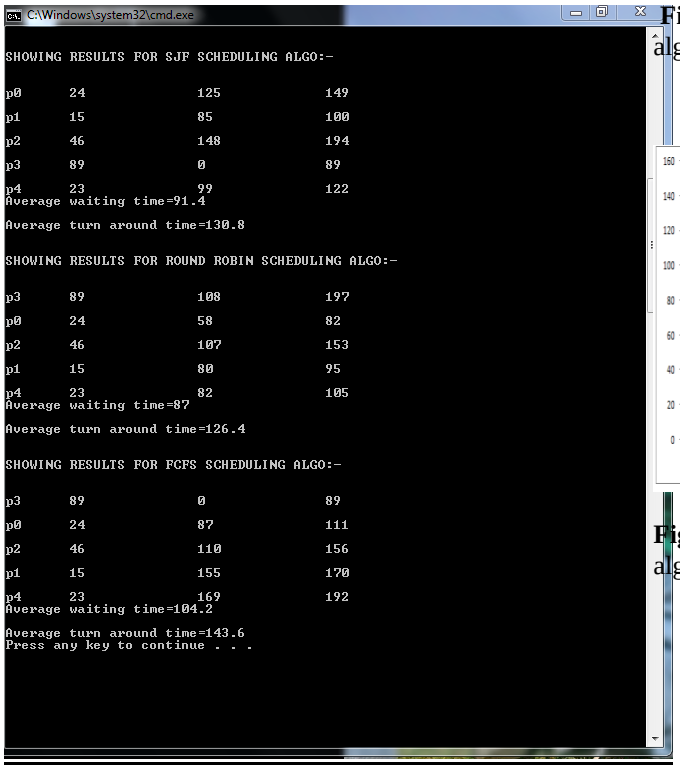




Example 3:-

Screenshot 5& 6





Bar Graphs of Average Turn Around Time of Different Algorithms

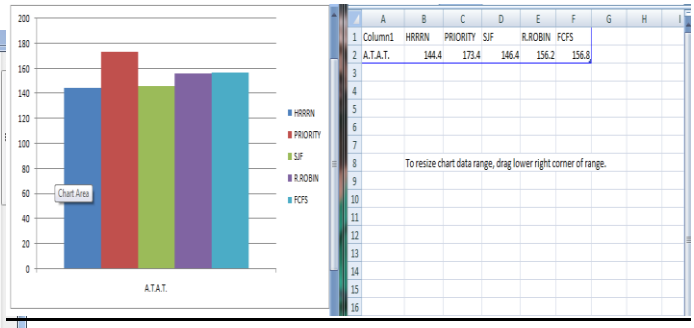


Figure 1: Average Turn around Time for different algorithms for example-1

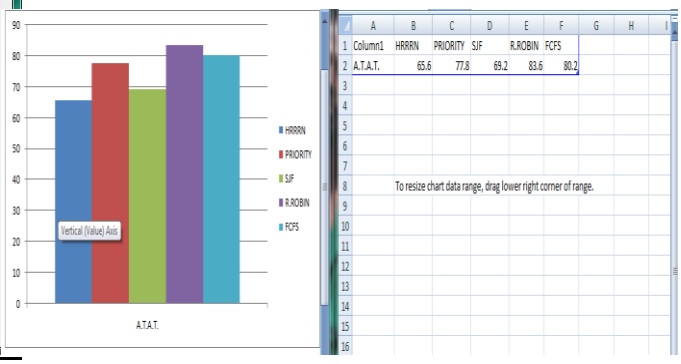


Figure 2: Average Turn around Time for different algorithms for example-2

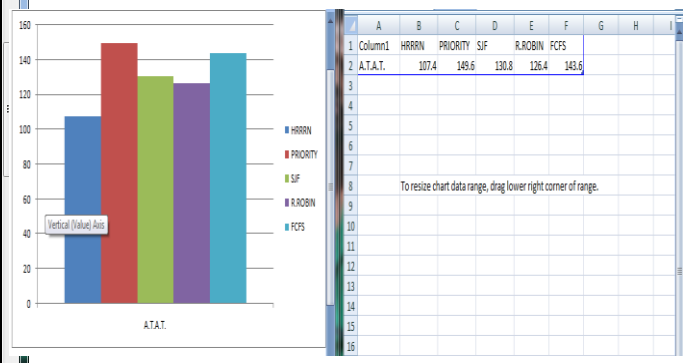


Figure 3: Average Turn around Time for different algorithms for example-3

Bar Graphs of Average Waiting Time Of Different Algorithms

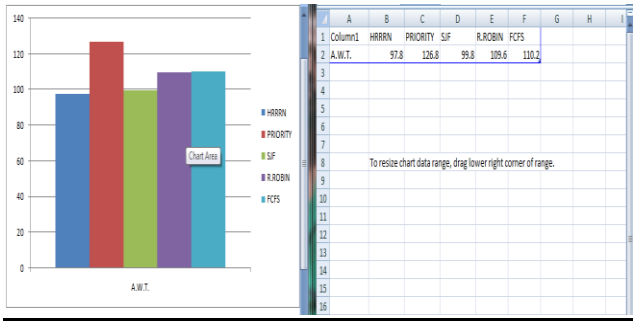


Figure 4: Average Turn around Time for different algorithms for example-1

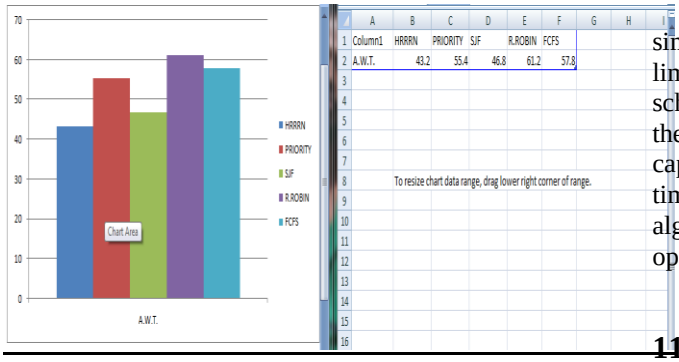


Figure 5: Average Turn around Time for different algorithms for example-2

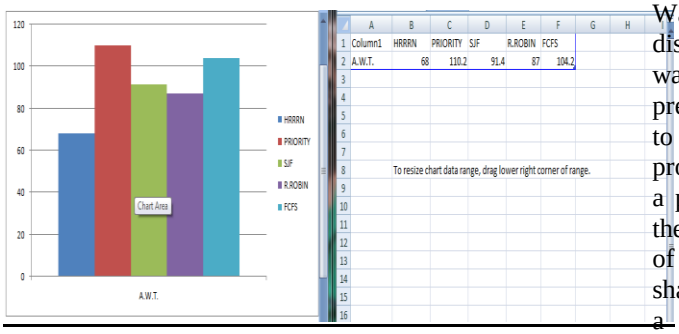


Figure 6: Average Turn around Time for different algorithms for example-3

Originally, such strategies were used in single CPU set-ups but more recently, they have been applied in networked, multi-CPU setups. Within this branch even, researchers have taken various approaches, each with their own goals and assumptions.

10. Future Scope

The work presented in this report can be expanded in many directions. Some of the directions are:

- Employing different performance criteria for comparison such as the makespan. The *makespan* is defined as maximum time needed to complete the execution of all the tasks arriving to the system .
- Applying scheduling technique on tasks that have dependencies among each other.
- Studying performance in real time applications where tasks have priorities and deadline constraints.
- Applying scheduling technique on distributed system. *A distributed system* is defined as a collection of independent computers that appear to the users of the system as a single computer.

It is recommended that any kind of simulation for any CPU scheduling algorithm has limited accuracy. The only way to evaluate a scheduling algorithm to code it and has to put it in the operating system, only then a proper working capability of the algorithm can be measured in real time systems. Hence in future the proposed algorithm will be implemented and can be tested in open source (LINUX).

11. Conclusion

Results have shown that the execution of FCFS produces 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, the preferred treatment of short processes in SJF tends to result in increased waiting times for long processes in comparison with FCFS. Thus, there is a possibility that long processes may get stuck in the ready queue because of the continuous arrival of shorter processes in the queue. RR achieves fair sharing of the CPU. Short processes execute within a single time quantum and thus exhibit good response time. It tends to subject long processes to relatively longer turn around and waiting times. In the case of priority-based scheduling there is a possibility that low priority processes will, in effect, be locked out by the higher priority ones. In other words, completion of a process within a finite time of its creation cannot be guaranteed with this scheduling policy. In RR there is fairness across the jobs, i.e. the jobs get equal time. However, upon completion of the time quantum, the PCB is linked to the tail of the ready queue and waits in the ready queue until it again gets the time quantum (after a

complete circle). The processes with very small burst lengths also wait for a long time in the ready queue. HRRN provides optimized results for turnaround time, response time and waiting time. There is no starvation for any jobs when using the HRRRN. The proposed algorithms HRRN is better in average turnaround time (as shown if figure 1, 2, 3). HRRN is also better in average waiting time (as shown in figure 4, 5, and 6).

11. References

Books

[1] A. Silberschatz, P. B. Galvin, G. Gagne, "Operating System Concepts", 7th ed., John Wiley & Sons, 2005.
[2] Milan Milenkovic, "Operating System Concepts and Design", Second Edition McGraw Hill International, 1992.
[3] Leland L. Beck, "System Software", 3rd Ed., Addison Wesley, 1997
[4] Milenkovic, M.(1992), *Operating System Concepts and Design*, McGraw Hill, International Edition.

[5] Silberschatz,A. and P.B. Galvin(1997) Operating System concepts, Fifth Edition, John Wiley & Sons, Inc.,

[6] Stallings,W., (1996) Computer Organization and Architecture; Designing for Performance, Fourth Edition, Prentice Hall

[7] Andrew S. Tanenbaum , and Albert S. Woodfhull(2005) , Operating Systems Design and Implementation,Second Edition.

Journals/Proceedings /Reports

[1] Patel J. & Solanki A.K. ,"CPU Scheduling: A Comparative Study" in the proceedings of the 5TH National Conference on Computing for Nation Development(INDIACom-2011) (supported by AICTE, CSIR) organized by Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi, (11th -12th March 2011),587-589, ISSN 0973-7529, ISBN 978-93-80544-00-7.
[2] C. Acosta et al., "The MPsim Simulation Tool," Technical Report UPC-DAC-RR-CAP-2009-15, Computer Architecture Dept., UPC, 2009.
[3] R.L. Arndt et al., "Method and Apparatus for Frequency Independent Processor Utilization Recording Register in a Simultaneously Multi-Threaded Processor," US Patent 7,870,406, to IBM Corp., Patent and Trademark Office, 2011.
[4] F.J. Cazorla et al., "Predictable Performance in SMT Processors: Synergy between the OS and SMTs," IEEE Trans. Computers, vol. 55, no. 7, pp. 785-799, July 2006.
[5] S. Eyerman and L. Eeckhout, "Per-Thread Cycle Accounting in SMT Processors," Proc. 14th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 133-144, 2009.
[6] S. Eyerman et al., "A Performance Counter Architecture for Computing Accurate CPI Components," Proc. 12th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 175-184, 2006.
[7] A. Fedorova, M. Seltzer, and M. Smith, "Improving Performance Isolation on Chip Multiprocessors via an Operating System Scheduler," Proc. 16th Int'l Conf. Parallel Architecture and Compilation Techniques (PACT), pp. 25-38, 2007.
[8] M.S. Floyd et al., "System Power Management Support in the IBM POWER6 Microprocessor," IBM J. Research and Development, vol. 51, no. 6, pp. 733-746, 2007.
[9] Standard Performance Evaluation Corporation, "SPEC CPU 2000 Benchmark Suite," http://www.spec.org, 2011.

Online References

[1] http://www.capricorn.org/~akira/cgi-bin/scheduler/index.html.

[2] http://www.utdallas.edu/~ilyen/animation/cpu/program/prog.html