Optimized Performance of CPU Scheduling

By

Akhtar Hussain

(2003-NUST-MS-Phd-CSE-189)

Submitted to the Department of Computer Engineering

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Software Engineering

Thesis Supervisor

Brig. Dr. Muhammad Younas Javed

College of Electrical & Mechanical Engineering

National University of Science & Technology

January 2009

ABSTRACT

CPU scheduling is the problem of deciding which computer process in the ready queue is to be allocated to the CPU for processing. It is a fundamental problem in operating systems in terms of minimizing the wait for the user when he or she simply wants to execute a particular set of tasks. There are many CPU scheduling algorithms, which solve this problem. Some of them are as FCFS, SJF, Round Robin, and Priority Algorithm etc.Different algorithms have different strengths, and no single one is ideal for every situation. Therefore, we need techniques for analyzing the characteristics of each algorithm in order to make an informed choice. Criteria for CPU scheduling include such as,

(i) Maximum CPU utilization under the constraints that the maximum response time is 1 second.

(ii) Maximum throughput such that turnaround time is linearly proportional to execution time.

(iii) Turnaround time is time taken by process to execute.

(iv)Waiting Time is the sum of the periods spent waiting in the ready queue.

(v) Response Time is the interval between submissions of a request until the first response is produced.

Once the selection criteria have been defined, there is a need to evaluate the various algorithms under consideration. There are number of different evaluation method. One of the evaluation method is called analytic evaluation. Analytic evaluation uses the given algorithm and the system workload to produce a formula or number that evaluates the performance of the algorithm for that workload. One type of analytic evaluation is deterministic modeling. This method takes a particular predetermined workload and defines the performance of each algorithm for that workload. In this thesis research work, different CPU scheduling methodologies (FIFO, SJF, Round Robin, and Priority) have been evaluated. A simulator have been developed which analyzes these algorithm based on Average Waiting Time (AWT). By using result of deterministic evaluator, an optimized solution for Round Robin is proposed for time sharing system. This Optimized solution has better performance than Existing Round Robin with respect to Average Waiting Time (AWT).

TABLE OF CONTENTS

Dedicated To

My all family members and friends for their continuous and never ending cooperation and encouragement.

Acknowledgement

First of all, I am greatly thankful to Allah, the Almighty, for enabling me to undertake and complete this enormous task. Without His help, it would have been impossible for me to achieve this milestone.

I express my deep gratitude to Brig. Dr. Muhammad Younas Javed, Head of Computer Engineering Department College of E &ME Rawalpindi NUST, to spare time from his extremely busy schedule, whenever I needed guidance. His constant supervision and advices have played a vital role in successful completion of this work.

Besides the above mentioned I am also thankful to all my beloved teachers at College of E&ME Rawalpindi NUST.

I can never forget to mention my friends Imran-Ullah Tariq, Basit Shahzad and Javed Iqbal who encouraged me a lot and gave me the moral and every other support whenever I needed.

Last, but not the least, I am grateful to all my family members specially my beloved Amma Gee, brothers, sisters and my beloved wife, who gave me moral and every other possible support Without their encouragement and prayers, it would have not been possible for me to achieve the position that Almighty Allah has bestowed on me today.

Akhtar Hussain

CHAPTER 1

SCHEDULING

Operating System

Operating Systems are resource managers. The main resource is computer hardware in the form of processors, storage, input/output devices, communication devices, and data. Some of the operating system functions are: implementing the user interface, sharing hardware among users, allowing users to share data among themselves, preventing users from interfering with one another, scheduling resources among users, facilitating input/output, recovering from errors, accounting for resource usage, facilitating parallel operations, organizing data for secure and rapid access, and handling network communications. An operating system may process its workload serially or concurrently. That is resources of the computer system may be dedicated to a single program until its completion, or they may be dynamically reassigned among a collection of active programs in different stages of execution. Because of their ability to execute multiple programs in interlaced fashion, such operating systems are often referred to as multiprogramming operating systems. Operating systems which support multiprogramming execute multiple programs concurrently. One of the objectives of multiprogramming is to maximize resource utilization, which is achieved by sharing system resources amongst multiple user system processes. Efficient resource sharing depends on efficient scheduling of competing processes.

1.1 Scheduling:

"Scheduling concerns the allocation of limited resources to tasks over time. It is a decision-making process that has as a goal the optimization of one or more objectives. [8]:

1.2 Objective of Scheduling

Many objectives must be considered in the design of a scheduling .In particulars, a scheduling discipline should:

- Be fair: If all tasks are treated the same and no task can suffer indefinite postponement, then such scheduling discipline will be fair.
- Maximize throughput: Scheduling discipline should complete maximum number of jobs per unit time.
- The response time of maximum jobs should be minimum.

- Be predictable: A given job should run in about the same amount of time and at about the same cost regardless of the load on the system.
- Minimize Overhead: Interestingly, this is not generally considered to be one of the most important objectives. Overhead is commonly viewed as wasted resources. But a certain portion of system resources invested as overhead can greatly improve overall system performance.
- Balance resource use: The scheduling mechanisms should keep the resources of the system busy. Process that will use underutilized resources should be favored.
- Achieve a balance between response and utilization. The best way to guarantee good response time is to have efficient resources available whenever they are needed. In real time systems, fast response time is essential, and resource utilization is less important.
- Avoid indefinite postponement: in many cases, indefinite postponement can be as bad as deadlock. Avoiding indefinite postponement is best accomplished by aging i.e. as a process wait for a resource, its priority should grow. Eventually, the priority will become so high that the process will be given the resource.
- Enforce priorities: In environment in which processes are given priorities, the scheduling mechanism should favor the higher priority process.
- Given preference to processes holding key resources: Even though a low priority process may be holding a key resource, the resource can be in demand by high priority processes. If the resource is non-preemptible, then the scheduling mechanism should give the better treatment than it would ordinarily receive so that the process will release the key resource sooner.
- Give better service to processes exhibiting desirable behavior.
- Degrade gracefully under heavy load: A scheduling mechanism should not collapse under the weight of a heavy system load. Either it should prevent excessive loading by not allowing new processes to be created when the load is heavy, or it should service the heavier load by providing a moderately reduced level of service to all processes.

As CPU is one of the main resources of Operating system. So to use this resource efficiently and effectively operating system must apply some scheduling mechanism. Such mechanisms are called CPU scheduling algorithm

1.3 CPU Scheduling:

When more than one process is run able, the Operating system must decide which to run first. The part of the operating system that makes this decision is called scheduler. And this mechanism is called CPU scheduling.CPU scheduling involves many algorithms [4].

Determining which processes run when there are multiple run able processes. Why is it important? Because it can have a big effect on resource utilization and the overall performance of the system.

Basic assumptions behind most scheduling algorithms:

- There is a pool of run able processes contending for the CPU.
- The processes are independent and compete for resources.
- The job of the scheduler is to distribute the scarce resource of the CPU to the different processes ``fairly'' (according to some definition of fairness) and in a way that optimizes some performance criteria.

There are several approaches for solving this problem. Some of them are as follows

- FCFS
- Shortest Job First
- Round Robin
- Priority Scheduling
- Shortest Remaining Time First Scheduling

Following are details of some scheduling algorithms.

1.3.1 First Come First Serve (FCFS)

The simplest of all the solutions (and the most naive) is the **first-come first-served** algorithm. With this scheme, whichever process requests CPU time first is allocated the CPU first. This is easily implemented with a FIFO queue for managing the tasks; as they come in, they're put on the end of the queue. As the CPU finishes each task, it pops it off the start of the queue and heads on to the next one.

The average waiting time for this technique, though, is often quite long, and that is the drawback of this approach. Let's take three processes that arrive at the same time in this order:

Process	CPU Time Needed (ms)
P-1	100
P-2	2
P-3	2

If we serve these guys up in order, we get this order of execution:

P-1 starts at time 0 and ends at time 100.
P-2 starts at time 100 and ends at time 102.
P-3 starts at time 102 and ends at time 104.

Thus, P-2 has to wait 100 milliseconds to start and P-3 has to wait 102 milliseconds. The average waiting time here is (0 + 100 + 102) / 3 = 67.3 milliseconds. But if we just switch the order a bit to P-2, then P-3, then P-1, then the average waiting time becomes (0 + 2 + 4) / 3 = 2 milliseconds. Clearly, the average waiting time under a purely first-in first-out system is going to often be poor if one task is significantly longer than the others [10].

1.3.2 Shortest Job First (SJF)

This, obviously, would be similar to the idea above, except that as each job comes in, it is sorted into the queue based on size. In concept, this idea makes a lot of sense, and one can easily prove that it is optimal in terms of giving the lowest average waiting time for a set of processes. The difficulty with this algorithm, though, knows which incoming process is indeed shorter than another. There is no way to know the length of the next CPU burst (the amount of time a given process will need), so this type of scheduling is largely impossible. It does have some use for scheduling long-term jobs, when we know exactly how long they will run and you are attempting to plan them out over a long period of time, but in terms of doing this on the fly with incoming processes, it is largely impossible. Length of next CPU burst is associated with each process. We use these lengths to schedule the process with the shortest time. There are two schemes in SJF.

- Non preemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.
- Preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).

1.3.3 Priority Scheduling

In this method each process has a priority associated with it, and as each process hits the queue, it is sorted in based on its priority so that processes with higher priority are dealt with first. This is essentially the shortest-job-first idea, except in that case the priority is the inverse of the time the process needs. A variation on this theme is **multilevel queue scheduling**, where each task is designated one of several different levels of importance, and the most important ones always run first. This is quite effective at scheduling jobs for a CPU, assuming that the operating system has the sense to properly assign priorities to the tasks. There is only one real problem with this method, and that itself has a clever solution. The problem occurs when the operating system gives a particular task a very low priority, so it sits in the queue for an exorbitant amount of time, not being dealt with by the CPU. If this process is something the user needs, there could be a very long wait. As a result, many operating systems use a technique called **aging**, in which a low priority process slowly gains priority over time as it sits in the queue. That way, no matter how low the priority of the process is, it will eventually be dealt with.

1.3.4 Round Robin Scheduling

Another approach to solving the CPU scheduling problem, **round-robin** scheduling, may be the ticket. In this approach, a **time slice** is defined, which is a particular small unit of time. In each time slice, the process being actively dealt with by the CPU runs until the end of the time slice; if that process is done, it is discarded and the next one in the queue is dealt with. However, if the process isn't done, it is halted and put at the back of the queue, and then the next process in line is addressed during the next time slice. This method vastly slows down short processes, because they have to share the CPU time with other processes instead of just finishing up quickly. Thus, a good rule of thumb is to make sure that the length of the time slice is such that 80% of your processes can run in one time slice. This way, only the longer processes wrangle over CPU time and the short jobs aren't stuck at the back of the queue. If we keep time slice large then it gives performance approximately equal to FIFO

1.3.5 Multiple-Processor Scheduling

CPU scheduling becomes more difficult when multiple processors are available. There are different rules in homogeneous and heterogeneous processors. Load sharing in applied to distribute the work among different processor so that equal amount of work can be done by all processor. There are two type of multiprocessing

- Symmetric multi processing: In this type of processing each processor makes its own scheduling.
- Asymmetric multi processing: In this type of processing only one processor can access data structure of system so master slave management is applied in this technique.

1.3.6 Real Time Scheduling

There are two type of real time system

- Hard real-time systems – These require to complete a critical task within a guaranteed amount of time.
- Soft real time systems They require that critical processes receive priority over less fortunate ones

There are several scheduling algorithms for real time systems [4].

Rate monotonic Algorithm: It assigns to each process a priority proportional to the frequency of occurrence of its triggering event. For example, a process to run every 20msec gets priority 50 and a process to run every 100 msec gets priority 10.At run time, the scheduler always runs the highest priority ready process, preempting the running process if need be.Liu and Layland proved that this algorithm is optimal.

Earliest Deadline First: Whenever an event is detected, its process is added to the list of ready processes. The list is kept sorted by deadline, which for a periodic event is the next occurrence of the event. The algorithm runs the first process on the list, the one with the closest deadline.

Least Laxity: It first computes for each process the amount of time it has to spare, called its laxity. If a process requires 200 msec and must be finished in 250 msec,its laxity is 50 msec.This algorithm chooses the process with the smallest amount of time to spare

While in theory it is possible to run a general-purpose operating system into real-time system by using one of these scheduling algorithms, in practice the context-switching overhead of general purpose systems is so large that real time-time performance can only be achieved for applications with easy time constraints. As a consequence, most real time work uses special real-time operating systems that have certain important properties. Typically these includes a small size, fast interrupt time, rapid context switch, short interval during which interrupts are disables, and the ability to mange multiple timers in the millisecond or microsecond range.

CHAPTER 2

CPU SCHEDULING

2.1 Process Concept

A process is most often defined as a program in execution. It is of two types, a user process or an operating system process. The CPU is assigned to processes by the operating system in order to perform computing work. In a single job operating system there is only one user process at a time. In a multiprogramming system ,however, there may be many independent processes competing for the control of CPU [1].For process management point of view operating system perform the following activities [2].

- Creating and removing process.
- Controlling the progress of processes, that is, ensuring that each logically enabled process makes progress towards its completion at a positive rate,
- Acting on exceptional conditions arising during the execution of a process, including interrupts and arithmetic errors.
- Allocating hardware resources among the processes.
- Providing a means of communicating messages or signals among processes.

A process is created when a user job begins execution and is destroyed when the job terminates Process is a dynamic concept that refers to a sequence of code in execution, undergoing frequent state and attribute changes. An executable program, on the other hand, is a static process template that may give rise to one or more processes. As an example of the distinction between these two concepts, consider execution of the text editor program in a multiprogrammed multi user system [1].The file containing the editor program in its executable form is the template for all editor processes. When a user invokes the editor, the operating system loads the editor program into the memory, creates an editor process, and schedules it for execution. The editor process then begins executing the editing commands issued by the user from the terminal. In so doing, the editor process uses a specific set of user designated input and output files. When editing session is complete the user issues some command to exit the editor process. Which then perform some housekeeping and terminates it self by calling operating system. At this the operating system closes the data files and erases the record of that specific instance of the editor process, which then ceases to exit.

2.1.1 Process States

A process goes through a series of discrete process states. Various events can cause a process to change states. The state of process is defined in part by the current activity of that process. Each process may be in one of the following states.

- New: The process is being created.
- Running: Instructions are being executed or a process is said to be in running state. if it currently has a CPU.
- Waiting: The process is waiting for some event to occur(such as an I/O completion or reception of a signal)
- Ready: A process is said to be in ready state if it could use a CPU if one were available or the process is waiting to be assigned to a processor.
- Terminated: The process has finished execution.

Process states are described in following Figure 2.1

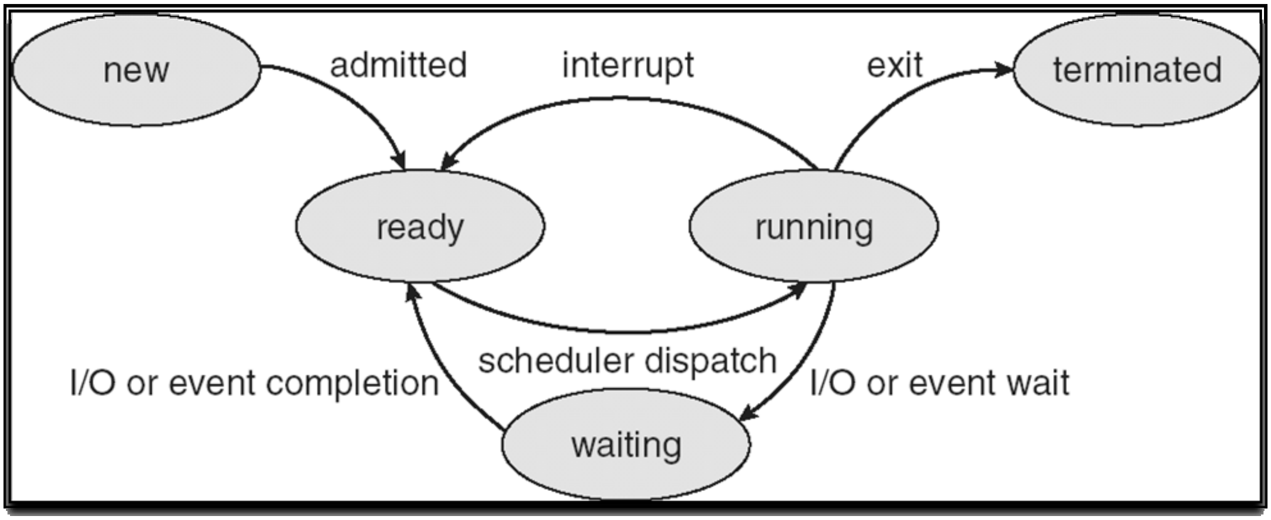


Figure 2.1 Process State Transition

These state names are arbitrary, and they vary across operating systems. The states that they represent are found on all systems, however. Only one process can be running on any processor at any instant, although many processes may be ready and waiting.

2.1.2 Process Control Block

The operating system groups all information that needs about a particular process into a data structure called a process descriptor or a Process control block (PCB).Whenever a process is created, the operating system creates a corresponding PCB to serve as its run time description during the lifetime of the process. When the process terminates, its PCB is released to the pool of free cells from which new PCBs are drawn. The PCB is a data structure with fields for recording various aspects of process execution and resource usage. Figure 2.2 show that information stored in PCB typically includes some or all of the following parameters [1].

- Process Name
- Process State
- Hard ware state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information
- File management information(open files, access rights)

Once constructed for a newly created process, the PCB is filled with the programmer defined attributes found in the process template or specified as the parameters of the appropriate operating system call to create process. Default values for other fields are usually assigned by the operating system at this time. For example, when a process is created hardware registers and flags are set to the values provided by linker/loader. Whenever that process is suspended, the content of the processor register are usually saved on the stack, and the pointer to the related stack frame is stored in the PCB,In this way ,the hardware state can be restored when the process is scheduled to run again

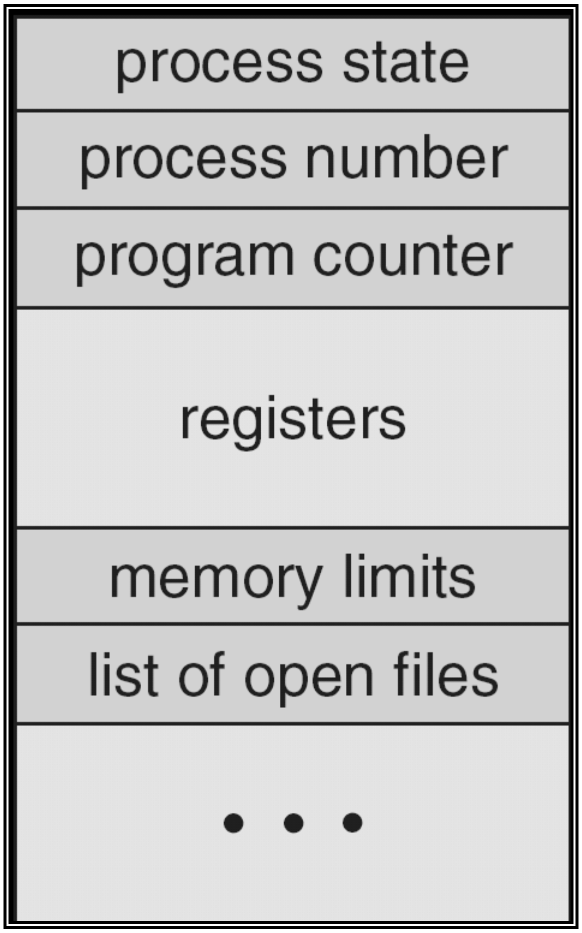


Figure 2.2 PCB

.

Blocked diagram of PCB is shown in Figure 2.2

2.2 Scheduling Queues

A state of a process is just a component of the global system state which encompasses all processes and resources. To keep track of all processes, the operating system maintains queues of processes classified by the current state of the related processes. In general there is a ready queue which contains process control blocks of all ready processes, and a suspended queue. By means of these queues operating system forms pools of processes in similar states that are likely to be examined by operating system resource allocation routines. For example the scheduler should search for the next process to run only in the ready queue [1].Three different types of queues, generally needed for process scheduling [2] are listed below.

2.2.1 Job Queue

As processes are submitted to the system, they are put into the queue, called Job queue. This queue consists of all processes residing on mass storage awaiting allocation of main memory.

2.2.2 Ready Queue

The processes that are residing in main memory and are ready and waiting to execute are kept on ready queue.

2.2.3 Device Queues

The list of processes waiting for a particular I/O device is called a device queue. Each device has its own device queue.

2.3 Types of Scheduling

In general, (job) scheduling is performed in three stages

Long-term (job) scheduling is done when a new process is created. It initiates processes and so controls the degree of multi-programming (number of processes in memory).Acting as the primary resource allocator, the long term scheduler admits more jobs when the resource utilization is low, and blocks the incoming jobs from entering the ready queue when utilization is too high.

Medium-term scheduling involves suspending or resuming processes by swapping (rolling) them out of or into memory. When the main memory becomes over-committed, the medium-term scheduler releases the memory of a suspended (blocked or stopped) process by swapping (rolling) it out

Short-term (process or CPU) scheduling occurs most frequently and decides which process to execute next. Short-term scheduler, also known as the process or CPU scheduler, controls the CPU sharing among the ‘ready’ processes. The selection of a process to execute next is done by the short-term scheduler. Usually, a new process is selected under the following circumstances

- When a process must wait for an event.
- When an event occurs (e.g., I/O completed, quantum expired).

- When a process terminates

In summary, both long and medium term schedulers control the level of multiprogramming and avoid (as much as possible) overloading the system by many processes and cause ‘‘thrashing’’. All level of scheduling is shown in Figure 2.3.The goal of short-term scheduling is to optimize the system performance, and yet provide responsive service. In order to achieve this goal, the following set of criteria is used [11]:

- CPU utilization
- I/O device throughput
- Total service time
- Responsiveness
- Fairness
- Deadlines

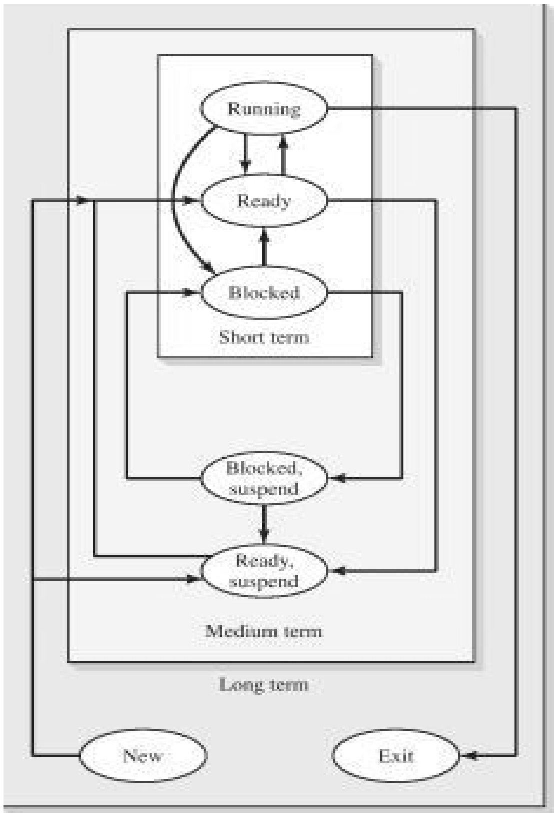


Figure 2.3 Level of Scheduler

2.4 CPU Scheduler

The CPU scheduler allocates the processor among the pool of ready processes resident in memory. Its main objective is to maximize system performance in accordance with the chosen set of criteria. Since it is in charge of ready to running state transitions, the CPU scheduler must be invoked for each process switch to select the next process to be run .In practice, the CPU scheduler is invoked whenever an event causes the global state of the system to change. Given that, any such change could result in making the running process suspended or in making one or more suspended processes ready, the CPU scheduler should be invoked to determine whether such significant changes have indeed occurred and if so, to select the next process to be executed. Some of the events that may cause rescheduling by virtue of their ability to change the global system state are:-

- Clock ticks (time base interrupts).
- Interrupts and I/O completions.
- Operating system calls.
- Sending and receiving of signals.
- Activation of interactive programs**.**

2.5 Scheduling Objectives

Many objectives must be considered in the design of a scheduling .In particulars, a scheduling discipline should:

Be fair: If all tasks are treated the same and no task can suffer indefinite postponement, then such scheduling discipline will be fair.

Maximize throughput: Scheduling discipline should complete maximum number of jobs per unit time.

The response time of maximum jobs should be minimum.

Be predictable: A given job should run in about the same amount of time and at about the same cost regardless of the load on the system.

Minimize Overhead: Interestingly, this is not generally considered to be one of the most important objectives. Overhead is commonly viewed as wasted resources. But a certain portion of system resources invested as overhead can greatly improve overall system performance.

Balance resource use: The scheduling mechanisms should keep the resources of the system busy. Process that will use underutilized resources should be favored.

Achieve a balance between response and utilization. The best way to guarantee good response time is to have efficient resources available whenever they are needed. In real time systems, fast response time is essential, and resource utilization is less important.

Avoid indefinite postponement: in many cases, indefinite postponement can be as bad as deadlock. Avoiding indefinite postponement is best accomplished by aging i.e. as a process wait for a resource, its priority should grow. Eventually, the priority will become so high that the process will be given the resource.

Enforce priorities: In environment in which processes are given priorities, the scheduling mechanism should favor the higher priority process.

Given preference to processes holding key resources: Even though a low priority process may be holding a key resource, the resource can be in demand by high priority processes. If the resource is non-preemptible, then the scheduling mechanism should give the better treatment than it would ordinarily receive so that the process will release the key resource sooner.

Give better service to processes exhibiting desirable behavior.

Degrade gracefully under heavy load: A scheduling mechanism should not collapse under the weight of a heavy system load. Either it should prevent excessive loading by not allowing new processes to be created when the load is heavy, or it should service the heavier load by providing a moderately reduced level of service to all processes.

2.6 Performance Criteria

Common performance measures and optimization criteria that scheduler may use in attempting to maximize system performance include [1, 3]:

- CPU utilization
- Throughput
- Turnaround time
- Waiting time
- Response time

The scheduler should also strive for fairness, predictability and repeatability so that similar behavior.

2.6.1 CPU Utilization

It is the average fraction of time during which the processor is busy. Being busy usually refers to the processor not being idle and includes both the time spent executing user programs and executing the operating system. The idea is to keep the CPU as busy as possible. In real systems it ranges from 40 percent for lightly loaded system to 90 percent for heavily loaded system. With CPU utilization approaching 100%, average waiting times and average queue lengths tend to grow excessively.

2.6.2 Throughput

Throughput is the amount of work that a computer can do in a given time period. It can be defined as amount of work completed in unit of time. One way to express throughput is by means of the number of processes executed in a unit of time. The higher the number, the more work is apparently being done by the system. In closed environments, such as batch processing systems, throughput can be a measure of scheduling efficiency. In systems, where service demands are generated by a large population of users, such as time sharing interactive systems, throughput is dictated by external factors and is a function of the request arrival rate and the processor service rate.

2.6.3 Turnaround Time

Turnaround time is defined as the time that elapses from the moment a program or job is submitted until it is completed by a system. It is the time spent in the system and may be expressed as sum of the execution time and waiting time. It is the sum of time periods spent waiting to get into the memory, waiting in the ready queue, executing on the CPU, and doing I/O.

2.6.4 Waiting Time

It is the time that a process spends waiting for resource allocation due to contentions with other processes. It is the sum of all waiting times a process experiences from the time of its creation to termination. In other words, waiting time is penalty imposed for sharing resources with others. Waiting time may be expressed as turnaround time minus actual execution time.

$W(x) = T(x) - x$

Where ‘x’ is the execution time(x) is the waiting time of the process requiring ‘x’ units of service, and T(x) is the process’s turnaround time.CPU scheduling does not really affect the amount of time during which a process executes or does I/O.It effects only the amount of time that a process spends waiting in the ready queue.So, for this work, the waiting time is assumed to be the total time a spends in the ready queue during its existence. As in batch processing system, there can be many processes concurrently executing, so the best way to assess their waiting time is to compute average waiting time in ready queue.

2.6.5 Response Time

In interactive systems, it is defined as the time from the submission of a request until the first response is produced. So the time elapsed from a moment the last character of a command line launching a program or a transaction is entered until the first result appears on the terminal will be called response time. It is desirable to maximize CPU utilization and throughput, and to minimize the turnaround time, waiting time and response time. In most of the cases average measures are optimized as in the case of average waiting time. However, some time it is desirable to optimize the minimum or the maximum values. As in the case of CPU utilization and throughput it is desirable to have maximum values, but to guarantee that all users get good service, maximum response time may be minimized.

2.6.6 Scheduler Design

A typical scheduler design process consists of selecting one or more primary performance criteria and ranking them in relative order of importance [1,3].The next step is to design a scheduling strategy that maximizes performance for the specified set of criteria while obeying the design constraints. Scheduler typically attempts to maximize the average performance of a system, relative to given criterion. However, due consideration must be given to controlling the variance and limiting the worst case behavior. For example, a user experiencing 10-second response time to simple queries has little consolation in knowing that the system's average response time is under 2 seconds.

One of the problems in selecting a set of performance criteria is that they often conflict with each other, as in the case with most engineering problems. The design of a scheduler usually requires careful balance of all the different requirements and constraints. With the knowledge of the primary intended use of a given system, operating system designer tend to maximize the criteria most important in given environment. For example, throughput and component utilization are the primary design objectives in a batch system.

2.7 Scheduling Disciplines

There are two type of scheduling discipline.

2.7.1 Non preemptive Scheduling

In batch processing non preemption implies that a selected job runs to completion or the running process retains ownership of allocated resources until it voluntarily surrenders control to the operating system. In other words, the running process is not forced to relinquish ownership of the processor when a higher priority process becomes ready for execution. However, when the running process becomes suspended as a result of its own action, say by waiting for an I/O completion, another ready process may be scheduled [1, 2, 3].

2.7.2 Preemptive Scheduling

With preemptive scheduling running process may be replaced by a higher priority process at any time without completing its assigned task. Similarly, it may relinquish control of CPU to another process after its allocated time slice expires. This is accomplished by invoking the scheduler whenever an event that changes the state of the system is detected. Since such events include a number of actions in addition to the voluntary surrender of control by running process, preemption generally necessitates more frequent execution of the scheduler. Thus, preemptive scheduling is generally more responsive. However, it imposes higher overhead since each process rescheduling entails a complete process switch [1, 2, 3].

2.8 CPU Scheduling Algorithms

CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU.There are many different CPU scheduling algorithms. In choosing which algorithm to use in a particular situation, properties of the various algorithms must be considered. For comparison of their performance, their performance criteria are compared. Popular algorithms are described in below mentioned sections [1, 2, 3]

2.8.1 First Come First Served (FCFS) Scheduling

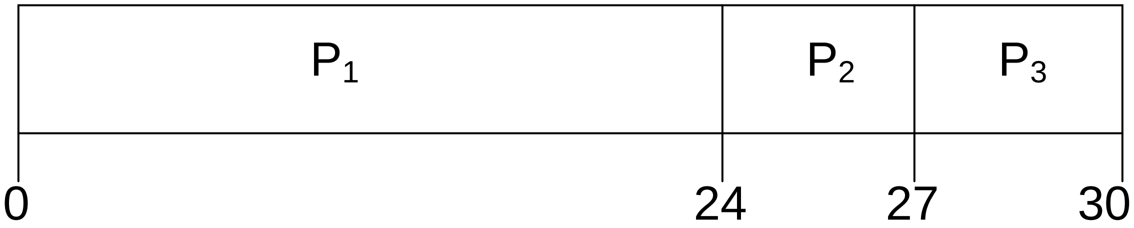
FCFS is by far the simplest scheduling discipline. The workload is simply processed in the order of arrival. Without preemption.Implementtation of the FCFS scheduler is quite straightforward. It can be easily managed with FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the ready queue. When CPU is free, it is allocated to the process at the head of the ready queue's running process is then removed from the ready queue. Because of its simplicity, the execution of FCFS results in smaller computational overhead [1,3].By failing to take into consideration the state of the system and the resource requirements of the individual scheduling entities,FCFS scheduling may result in poor performance, lower throughput and longer average waiting times. Short jobs may suffer considerable turnaround delays and waiting times when one or more long jobs are in the system.

The average waiting time under the FCFS policy, however, is often quite long. Consider the following set of polices that arrive at time 0, with the length of the CPU burst time given in milliseconds:

Process	**Burst Time**
P1	24
P2	3
P3	3

Suppose that the processes arrive in the order: P_1, P2, P_3

The Gantt Chart for the schedule is:



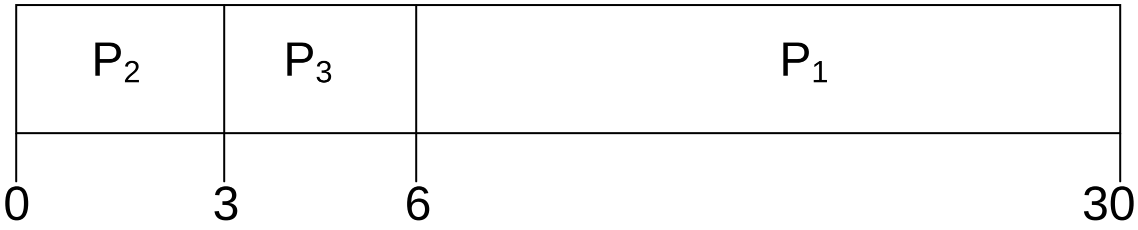
Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$

Average waiting time: (0 + 24 + 27)/3 = 17

Suppose that the processes arrive in the order

P_2, P_3, P_1.

The Gantt chart for the schedule is:



Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$

Average waiting time: (6 + 0 + 3)/3 = 3

Much better than previous case.

Convoy effect short process behind long process

2.8.2 Shortest Job First (SJF) Scheduling

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. The shortest job first scheduling favors' the process/job with the smaller burst time. This algorithm associate with each process the length of the latter's next CPU burst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If the two processed have the same CPU burst, FCFS is used to break the tie. A more appropriate term would be shortest next CPU burst, because the scheduling is done by examinining the length of the next CPU burst of the process, rather then its total length. The SJF scheduling is provably optimal, in that it gives the minimum average waiting time for a given set of processes. By moving a short process before a long one, the waiting time of the short process decreases more than it increases the waiting time of the processes. Consequently, the average waiting time decreases.

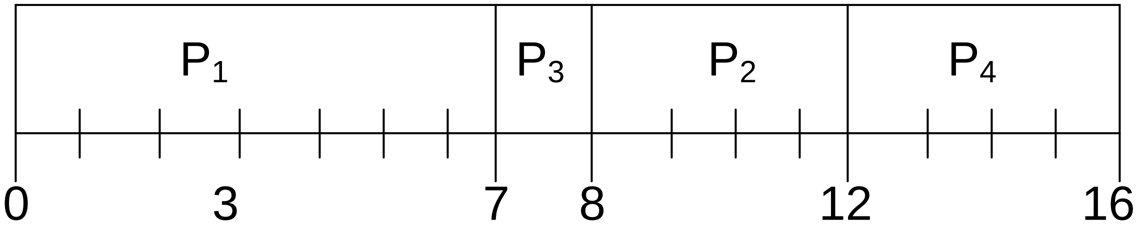
There are two schemes with SJF.

Non-Preemptive: once CPU given to the process it cannot be preempted until completes its CPU burst. Although the SJF non preemptive is optimal, it cannot be implemented at the level of short-term scheduling. There is no way to know the length of the next CPU burrstone approach is to try to approximate SJF non-preemptive scheduling. We may not know the length of next CPU burst, but we may be able to predict its value. We expect that next CPU burst will be similar in length to the previous ones. As an example considers the following set of processes, with the length of the CPU burst time given in milliseconds:

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

SJF (non-preemptive)



Average waiting time = (0 + 6 + 3 + 7)/4 = 4

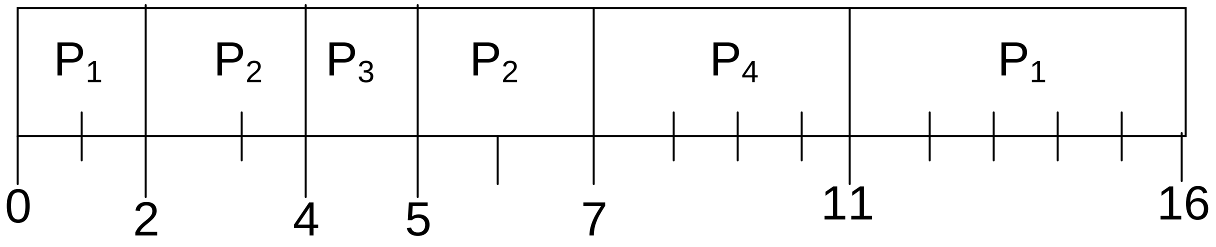
Preemptive If a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF). Shortest job first preemptive is very useful for time sharing. In this, the process with the smallest estimated run-time to completion is run in next, including new arrivals. In SJF non-preemptive, once job begins execution, it runs to completion. In Shortest remaining first (SRT)
Running process may be preempted by new process with shorter estimated run time. Again SRT require estimates of the future to be effective, and the designer must provide for potential user abuse of system scheduling strategies.SRT has higher overhead than SJF non-preemptive. It must keep track of the elapsed service time of running job, and must handle occasional preemptions/Arriving small processes will run almost immediately. Longer jobs however, have

an even longer mean waiting time and variance of waiting times than in SJF non-preemptive. As an example

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

SJF (preemptive)



Average waiting time = (9 + 1 + 0 +2)/4 =3ms

Determining Length of Next CPU Burst

Length of next CPU burst can be find by using the length of previous CPU bursts, using exponential averaging. Formula for finding the next burst is as follows [2]

$$\tau_{n=1} = \alpha\, t_n + (1-\alpha)\tau_n.$$

Examples of Exponential Averaging

Let $\alpha = 0$

Then $\tau_{n+1} = \tau_n$

Recent history does not count.

$\alpha = 1$

$\tau_{n+1} = t_n$

Only the actual last CPU burst counts.

If we expand the formula, we get:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \alpha\, t_n\, -1 + \ldots$$

$$+(1 - \alpha)^j\, \alpha\, t_n\, -1 + \ldots$$

$$+(1 - \alpha)^{n=1}\, t_n\, \tau_0$$

Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor.

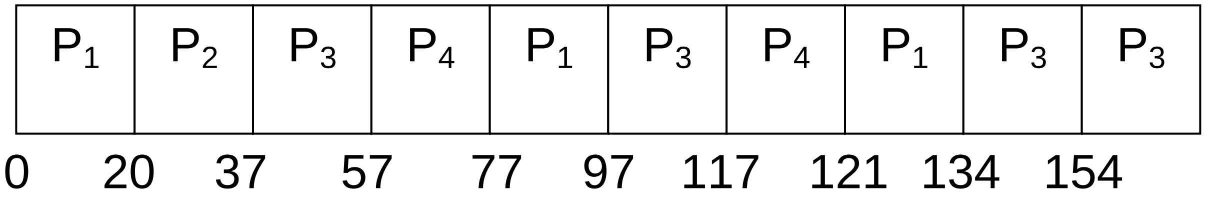
2.8.3 Round Robin (RR) Scheduling

It is one of the oldest, simplest, and fairest and most widely used scheduling algorithms, designed especially for time-sharing systems. A small unit of time, called time slices or quantum is defined. All run able processes are kept in a circular queue. The CPU scheduler goes around this queue, allocating the CPU to each process for a time interval of one quantum. New processes are added to the tail of the queue. The CPU scheduler picks the first process from the queue, sets a timer to interrupt after one quantum, and dispatches the process. If the process is still running at the end of the quantum, the CPU is preempted and the process is added to the tail of the queue. If the process finishes before the end of the quantum, the process itself releases the CPU voluntarily. In either case, the CPU scheduler assigns the CPU to the next process in the ready queue. Every time a process is granted the CPU, a **context switch** occurs, which adds overhead to the process execution time. If there are **n** processes in the ready queue and the time slice is **q**, then each process ideally would get 1/nof the CPU time in chunks of **q** time units, and each process would wait no longer than **nq** time units until its next quantum. A more realistic formula would be $n(q + o)$, where **o** is the context switch overhead. So, for practical purposes, it is desirable that the context switch be negligible compared to the time slice. The performance of the Round-Robin algorithm depends heavily on the size of the quantum. If the quantum is very large, the Round-Robin algorithm is similar to the First-Come, First-Served algorithm. If the quantum is very small, the Round-Robin approach is called processor sharing. Round Robin is effective in time sharing environments in which the system needs to guarantee reasonable response times of interactive users.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 quanta and thus be forced to cycle through the ready queue a little time before completion.RR scheduling tends to subject long processes to relatively long turnaround and waiting time. Such processes, however, may best be run in the batch mode [1].Implementation of RR scheduling requires support of an interval timer. The timer is usually set to interrupt the operating system whenever a time slice expires and thus forces the scheduler to be invoked. The scheduler itself simply stores the context of the running process, moves it the end of the ready queue and dispatches the process at the head of the ready queue. The Scheduler is also invoked to dispatch a new process whenever the running process surrenders control to the operating system before expiration of the time quantum, say, by requesting I/O operation. The interval timer is usually reset at that point, in order to provide the full time slot to the new running process. The preemption overhead is kept low by effective context switching mechanisms and by providing adequate memory for the processes to reside in main storage at the same time. The average waiting time under the round robin policy, however, is often quite long [9].

Example: RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

The Gantt chart is:



Typically, higher average turnaround than SJF, but better response. The relationship between the time slice and performance is nonlinear. Reduction of the time slice should not be carried too far in anticipation of better response time. Too short a time slice may result in significant overhead due to the frequent timer interrupts and process switches which causes context switches. [1].

Figure 2.4, 2.5 shows the way in which smaller time quantum increases context switches.

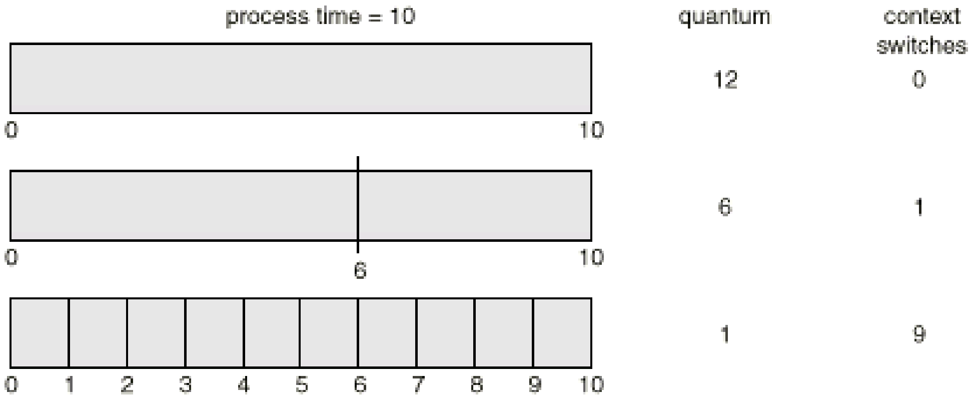
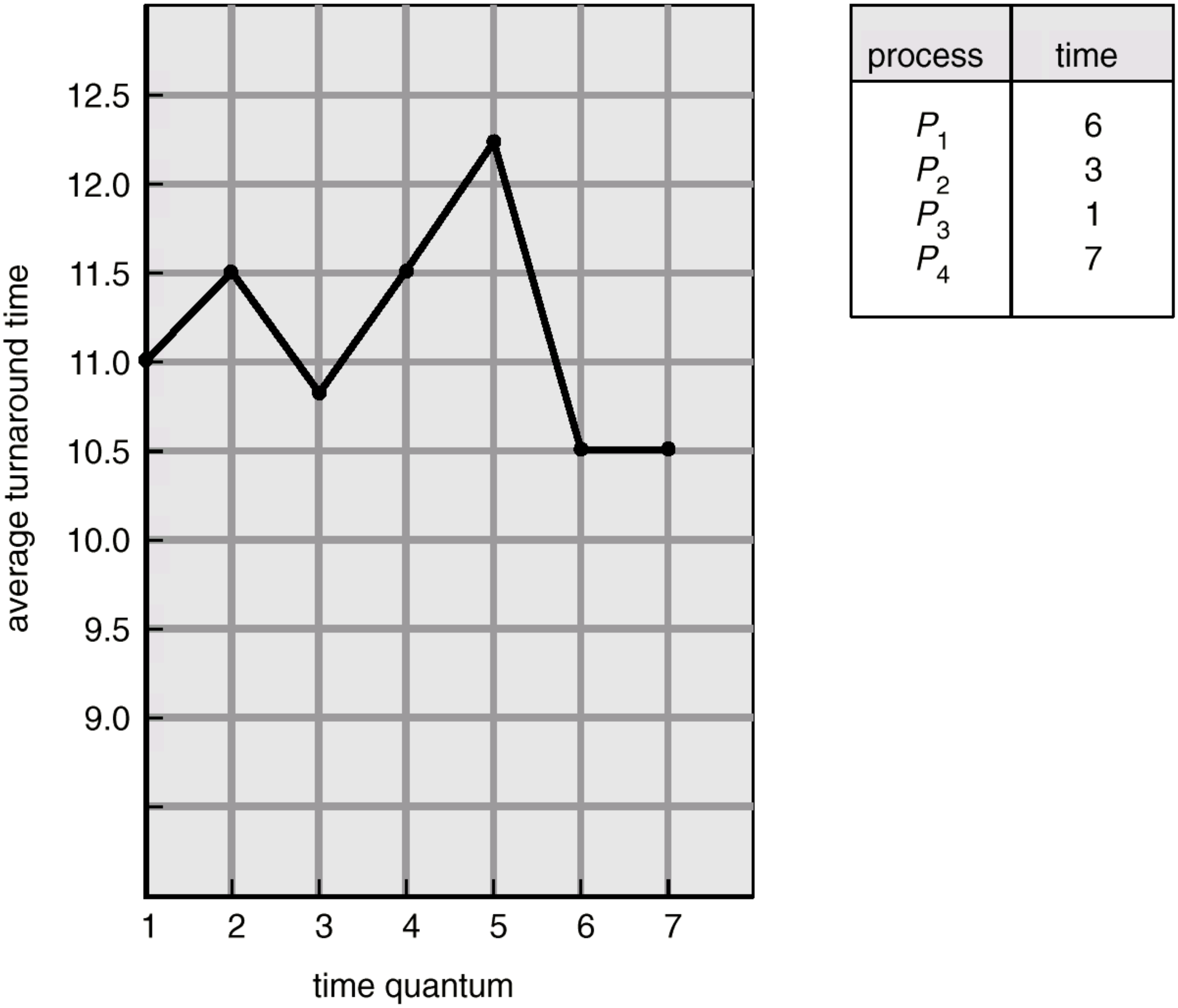


Figure 2.4 Turnaround Time Varies With the Time Quantum

Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Figure 2.5 Turn around Time vs. Time quantum

Round Robin scheduling is often regarded as “fair” scheduling discipline. The performance of RR scheduling is very sensitive to the choice of the time slice that is usually on the order of magnitude of milliseconds. This allows thousands of instructions to the running process per quantum. The instruction per slice measure is more suitable for comparing different systems because the time duration measure does not reflect the fact that processors with different speeds may accomplish different amounts of work within a time slice of a given duration. In summary, RR is primarily used in time-sharing and multiuser systems where terminal response time is important [1, 3].

2.8.4 Priority Scheduling

In priority scheduling, processes are allocated to the CPU on the basis of an externally assigned priority. The key to the performance of priority scheduling is in choosing priorities for the processes. Priority scheduling may cause low-priority processes to starve. This starvation can be compensated for if the priorities are internally computed. Suppose one parameter in the priority assignment function is the amount of time the process has been waiting. The longer a process waits, the higher its priority becomes. This strategy tends to eliminate the starvation problem. Run highest-priority processes first, use round-robin among processes of equal priority. Re-insert process in run queue behind all processes of greater or equal priority.

- Allows CPU to be given preferentially to important processes.
- Scheduler adjusts dispatcher priorities to achieve the desired overall priorities for the processes, e.g. one process gets 90% of the CPU.

2.8.5 Multiple-Level Queues (MLQ) Scheduling

The Scheduling disciplines described above are more suited to particular applications. But in a system having variety of jobs running concurrently, for example jobs with time critical events, some interactive jobs, and a few long non interactive jobs. Good performance can not be guaranteed if the system supports only one scheduling policy. To solve this problem, one approach is to combine several scheduling disciplines in one system. For example operating-system processes and device interrupts may be subjected to preemptive priority scheduling, interactive programs to RR scheduling, and batch jobs to FCFS or SJF [1, 3].Multiple ready queues are maintained each with a different scheduler as shown in Figure 2.6. The workload is classified according to its characteristics and is entered into an appropriate ready queue. Division of the workload might be into following classes.

- High priority system process
- Interactive programs
- Batch jobs
- These classes may be subdivided into subclasses. For example system processes can have two subclasses, such as:-

- Service of hardware are high priority interrupts
- Services of system calls or traps.

If the hardware interrupts have higher priority then they may have a separate ready queue. A process may be assigned to specific queue on the basis of its attributes that may be user or system supplied. Each queue may then be serviced by the scheduling discipline must also be devised for scheduling between queues. For example, the processes from the highest priority queue are serviced until that queue becomes empty. This scheduling discipline may be preemptive priority type. When the highest priority queue becomes empty the next queue may be serviced using its own scheduling discipline (e.g. RR for interactive processes).Finally, when both higher priority queues become empty, a batch process may be selected from the lowest priority queue. A process in a lower priority queue may be preempted by arrival of a process in one of the higher priority queues. This discipline maintains responsiveness to external events and interrupts at the expense of frequent preemptions [1, 3].

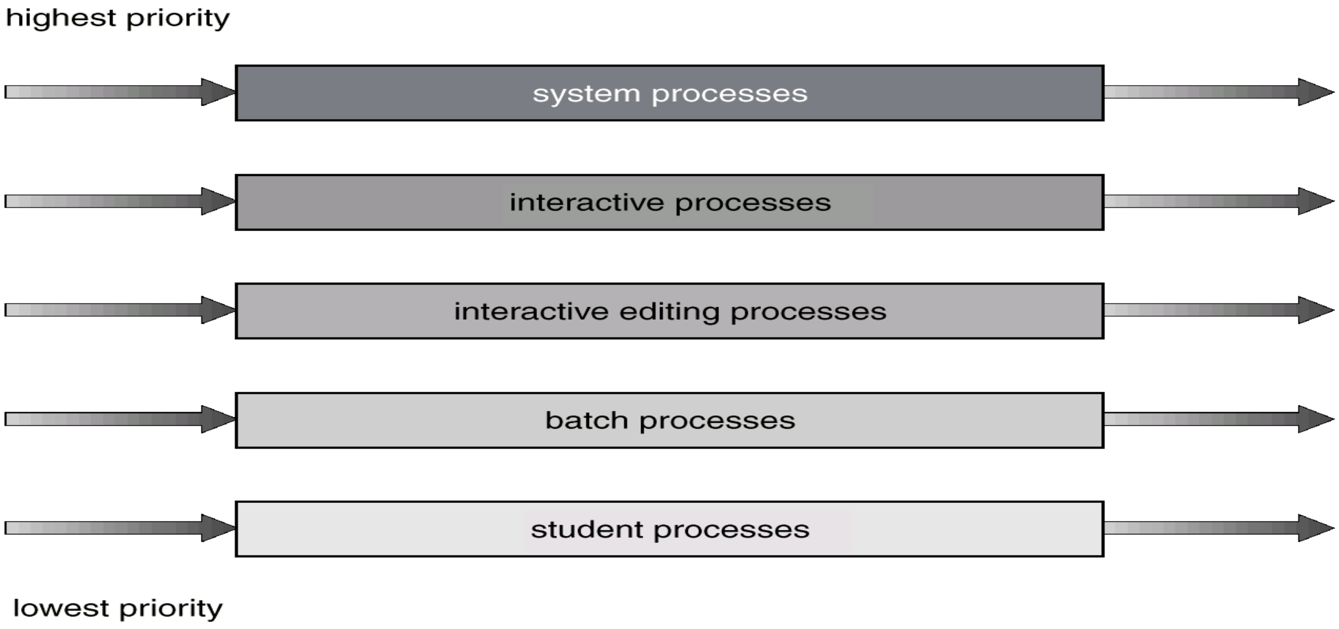


Figure 2.6 Multi level Queues

2.8.6 Multiple level Feedback Queues Scheduling

Multiple queues in a system may be used to increase the effectiveness and adaptivness of scheduling in the form of multiple-level queues with feedback. Rather than having fixed class of processes allocated to specific queues, the idea is to make traversal of a process through the system dependent on its run-time behavior. For example each process may start at the top level queue. If the process is completed with a given time slice it departs the system. Processes that need more than one time slice may be reassigned by the operating system to a lower priority queue that gets a lower percentage of the processor time. If the process is still not finished after having run a few times in that queue it may be moved to yet another, lower-level queue. The idea is to give preferential treatment to short processes and have the longer ones slowly move into lower-level queues [1, 3].Figure 2.7 shows the working of multilevel feed back queues.

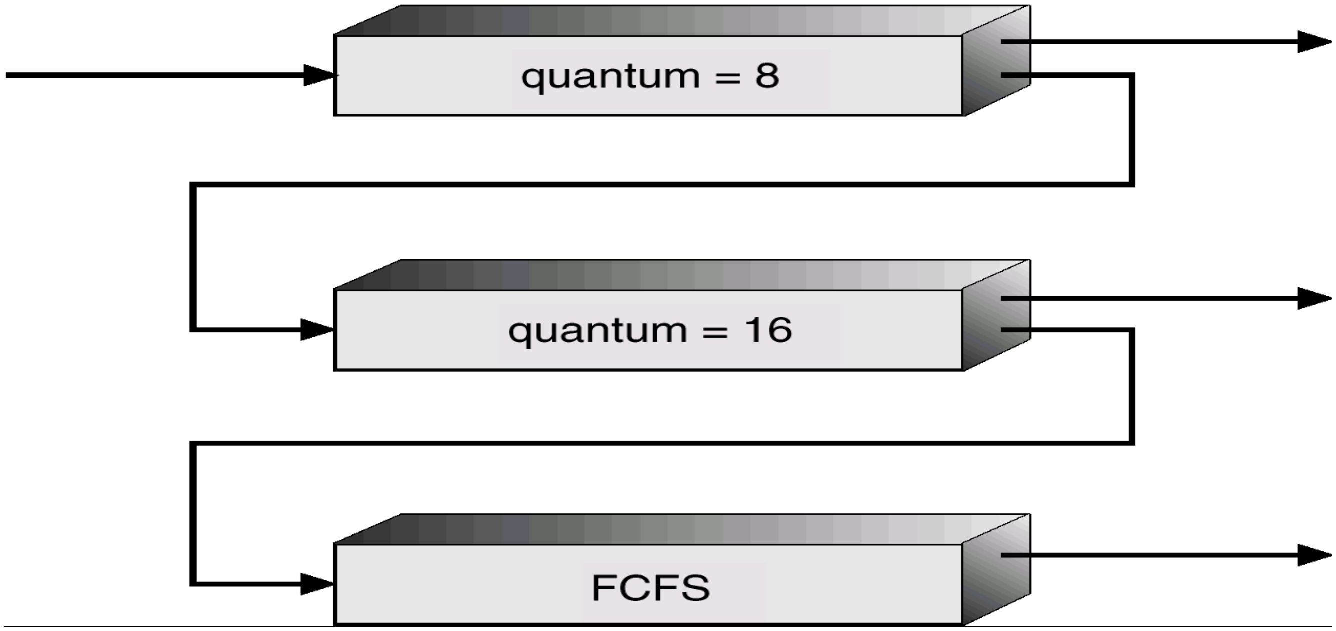


Figure 2.7 Multi level Queues

2.9 CPU Scheduling in different operating System

2.9.1 UNIX

UNIX SVR4 introduces the notion of priority classes and supports two types of them, although the design is flexible enough to incorporate new priority classes into the scheme as they become available. The user priority classes officially supported, along with the System-class for system kernel processes only, provide support for time sharing-processes (similarly to all previous releases of UNIX), and for real-time processes. A dynamic process selection scheme is used to determine which process will get the CPU next. All processes in the SVR4 belong to a particular process priority class. Each class supported by the system has an associated set of class dependent routines to calculate the priority level of a process and determine on which priority queue it is to be placed. These routines decide how a process is scheduled to run. Instead of a single run queue (a.k.a. ready queue), like in older UNIX implementations, there are now multiple priority queues, one for each priority value used by the system. The kernel then makes use of class independent routines to select a process from the highest valued priority queue, as opposed to the lowest used in previous implementations. In general, a process in the time-shared priority class is subject to a round-robin approach similar to the one of previous implementations: every process is eventually given use of the CPU, although some processes may get it before others, depending on their operating characteristics. However, a real-time process is given a higher precedence and is guaranteed to be selected to run before any time-sharing process. To this aim many preemption points are placed throughout the kernel, which can thus be interrupted. This is a very remarkable difference with respect to the older, time-sharing only, implementation, in which a process might never be preempted while operating in kernel mode. The kernel maintains a priority dispatch queue for each priority value in the system (all integer values in a pre-defined range) and they are held together in an array of dispatch queues. A scheme of this arrangement is shown in fig. 2,8. Each element of a queue is a *proc* structure, i.e. the part of the process control block (the ``user area", in UNIX parlance, as we saw previously) that contains the scheduling-related information. The queues in the array are sorted by priority value, while the element of each queue is managed in round-robin fashion. Priority class data are represented within each proc structure, and proc structures themselves change queue dynamically. The scheduling algorithm is then simply to always select the first process of the dispatch queue with highest priority. However, processes in the time-shared priority class have variable priorities computed at runtime (similarly to 4.2BSD), hence ``move" from queue to queue, while real-time (and system) processes have fixed priorities (unless the user changes them explicitly).

A real-time process can literally take-over the machine: while there's a real-time process on a dispatch queue no other system or time-shared process is scheduled. Thus, a real time process always has a higher priority than a system or time-shared process. Hence it is the programmer's responsibility to configure them so that the kernel is given the time it needs for normal system operation (i.e. paging, swapping, servicing time-shared processes,)

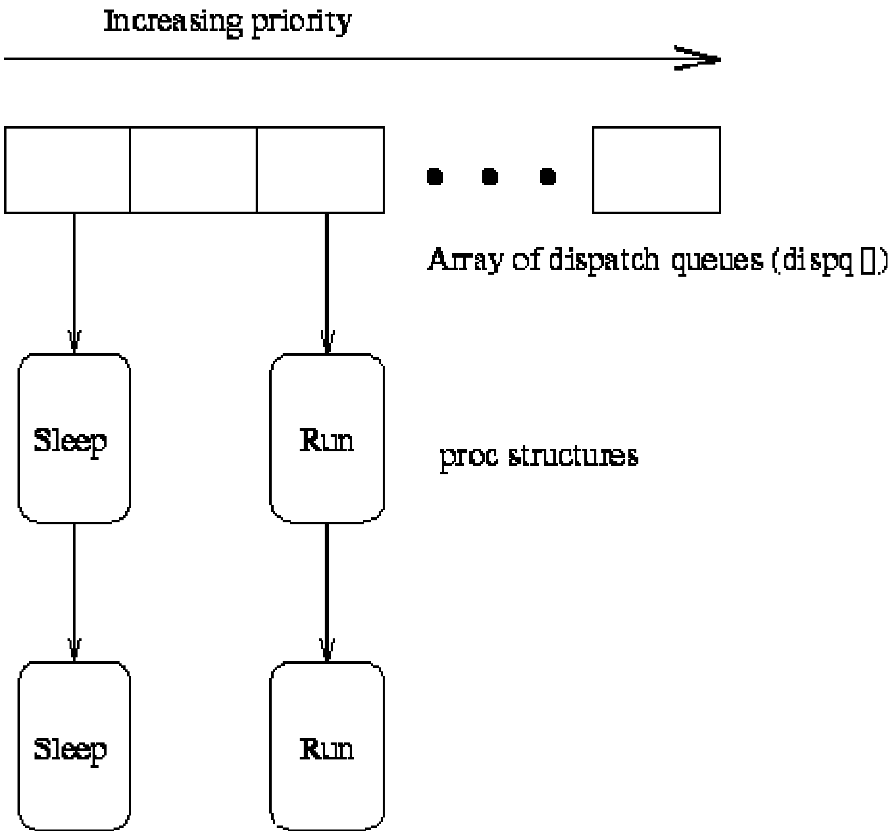


Figure 2,8: Priority queues data structure.

Priority class data are represented within each proc structure, and proc structures themselves change queue dynamically.

The scheduling algorithm is then simply to always select the first process of the dispatch queue with highest priority. However, processes in the time-shared priority class have variable priorities computed at runtime (similarly to 4.2BSD), hence ``move'' from queue to queue, while real-time (and system) processes have fixed priorities (unless the user changes them explicitly). A real-time process can literally take-over the machine: while there's a real-time process on a dispatch queue no other system or time-shared process is scheduled. Thus, a real time process always has a higher priority than a system or time-shared process. Hence it is the programmer's responsibility to configure them so that the kernel is given the time it needs for normal system operation (i.e. paging, swapping, servicing time-shared processes,...) [12]

2.9.2 Windows Vista

The CPU scheduling algorithm has been modified to make scheduling much more precise: on supported CPUs, an instruction-level precision clock will be used to precisely record how much CPU time is consumed by a thread. Also, the scheduling algorithm has also been modified to allow fair scheduling. This makes a CPU intensive program much less likely to affect overall responsiveness. Experiment: run a dead loop program and see rosh completion speed. In this way, Windows is comparable to Linux with the 2.6.24 kernel CFS (completely fair scheduler).

The scheduler was modified in Windows Vista to use the cycle counter register of modern processors to keep track of exactly how many CPU cycles a thread has executed, rather than just using an interval-timer interrupt routine

CHAPTER 3

Design

and

Implementation

3.1 Problem with CPU Scheduling Algorithms

In this section the problems and deficiencies with existing algorithms have been analyzed.

3.1.1 Problem with FCFS Scheduling

As in FCFS priority is given to the process which has arrived first. Here is an example of taking a set of some processes such that First process which arrives has large burst times, then the other process arrives which has small burst times. The CPU starts executing the processes which arrives first. The processes with large CPU burst get the maximum time of the CPU.This generates the starvation for the processes with small CPU bursts, which are at the bottom of the queue. So the FCFS scheduling is non preemptive. Once the CPU has been allocated to a process, that process keeps the CPU, until it is not completed. The FCFS algorithm is particularly troublesome for time sharing systems, where each user needs to get a share of the CPU at regular interval. It is disastrous to allow one process to keep the CPU for an extended period of time.

3.1.2 Problem with Shortest Job First (SJF) Scheduling

The Shortest Job First is optimal, but it cannot be implemented at the level of short term CPU scheduling. There is no way to know the length of the next CPU burst. The Shortest Job First is that it requires precise knowledge of how long a job or process runs, and this information is not usually available. The best SJF can do is to rely on user estimates of run times. In production environments where the same jobs run regularly, it may be possible to provide reasonable estimates. But in development environment users rarely know how long their program executes. The non preemptive SJF scheduling is not useful in timesharing environment in which the response time must be guaranteed. The SJF preemptive has high overhead than SJF non preemptive. It must keep track of the elapsed service time of the running job, and must handle occasional preemption. Arriving small processes run almost immediately. Longer Job ,however, have an even longer mean waiting time and variance of waiting time in SJF.So theoretically ,shortest job preemptive offers minimum waiting times. But because of preemption overhead it is possible that SJF might actually perform better in certain situations.

3.1.3 Problem with Priority Scheduling

A major problem with priority scheduling algorithm is indefinite blocking (or starvation).A process that is ready to run but lacking the CPU can be considered blocked, waiting for the CPU.A priority scheduling algorithm leaves some low priority processes waiting indefinitely for the CPU. In a heavily loaded computer system, a steady stream of higher priority process prevents a low priority process from ever getting the CPU.The priority scheduling preemptive has high overhead than priority scheduling non preemptive. It must keep track of the priority of the running job, and must handle occasional preemption. Arriving processes with high priority w run almost immediately. Low priority jobs, however, have an even longer mean waiting time.

3.1.4 Problem with Round Robin Scheduling

The Round Robin scheduling is the preemptive version of FCFS.The processes are treated on the basis of FCFS, but are preempted after a fixed time quantum. The measuring of time quantum is difficult. Because, if the quantum time is too small, then the overhead of preemption increases and if the time quantum is too large then it works same as the FCFS.So the optimized quantum time is necessary. The other overhead in Round Robin is to keep the record of all the processes where they are preempted. The state of all the processes is saved where it stops the execution. The other major problem with Round Robin is that; for example if there is a heavily loaded system, the quantum time is 22 milliseconds, if there is a process of 24 milliseconds, the process is preempted after 22 milliseconds only 2 milliseconds for the process are left. This process of 2 milliseconds gets time after the completion of whole queue. So the waiting time for this 2millisecond processes are very large. There is no starvation for this process but the waiting time is too long.

3.2 Proposed Algorithm

The main objective of this research work is to optimize the performance of CPU Scheduling using Effective Methodologies for Time Sharing Systems. As Round Robin is much suitable for time sharing systems. Round-Robin (RR) is one of the simplest scheduling algorithms for processes in an operating system, which assigns time slice to each process in equal portions and in order, handling all processes without priority. Round Robin scheduling is both simple and easy to implement, and starvation-free. Round-Robin scheduling can also be applied to other scheduling problems, such as data packet scheduling in computer networks. In this algorithm the main objective is to minimize the average waiting time, turn around time and response time of Round Robin algorithm..This new algorithm has been evaluated under following considerations.

- Maximize CPU utilization
- Minimize Average waiting time
- Minimize throughput such that turnaround time is linearly proportional to total execution time.

3.2.1 Optimized Round Robin Algorithm

This Optimized Round Robin algorithm is enhanced policy to maximize the throughput and to minimize waiting time and response time. Optimized Round Robin is effective in time sharing environments in which the system needs to guarantee reasonable response times to interactive users. The preemption overhead is kept low by effective context switching mechanism and by providing adequate memory for the processes to reside in main storage at the same time.

3.2.2 Basic Theory

The main features of Round Robin are:

.Optimized Round Robin is based on existing Round Robin. The technique it uses is based upon Round Robin. It optimizes the performance of Round Robin. Average Wait Time (AWT) is reduced by this technique, it also reduces response and Turn around time. The algorithm of Optimized Round Robin is as follows:

(1) Break the process in to segment according to time quantum.

(2) Sort segments of a processes so as smallest comes first.

(**3)**Sort the whole ready queue so that the process with smallest segment comes at the head of the ready queue.

(**4)**Now CPU starts executing the processes from head of the queue.

(5) During execution if a new process comes, it is adjusted accordingly.

3.2.3 Explanation

The Optimized Round Robin scheduling sorts all jobs according to their arrival time. Each Job is divided into small segments the size of each segment is equal or less than time quantum .All segments of each process are sorted according to time quantum. Then processes are again sorted according to segment. Sort the whole ready queue so that the process with smallest segment comes at the head of the ready queue. Now CPU starts executing processes according to Round Robin.For further explanation an example of 5 processes is considered ,such that all processes arrive at same time i.e. at 0 ms.Suppose time quantum is 10 ms.In first step process are sorted according FIFO and the all processes are divided into segments according to time quantum.

Step 1

P1 = 10 => 10

P2= 29 => 10,10,9

P3 = 3 => 3

P4 = 7 => 7

P5 = 12 => 10,2

In second step all segments within a process are sorted according to SJF.

Step 2

P1 = 10

P2 = 9, 10,10

P3 = 3

P4 = 7

P5 = 2,10

In third step processes are sorted with respect to segment length, such that process with smallest segment is served first.

Step 3

P_5=2, 10

P_3=3

P_4 = 7

P_2 = 9,10,10

P_1 = 10

Then all processes execute according to Round Robin fashion. Gant chart for above processes is shown as:

P5	P3	P4	P2	P1	P5	P2

0 2 5 12 21 31 41 61

Wait Time of P1 = 21

AWT = $\underline{21+32+2+5+29}$ = $\underline{89}$ = 17.8 ms

Wait Time P_2 = 12+20=32

Wait Time of P_3 = 2

Wait Time of P_4 = 5

Wait of P_5 = 29

Where as if above processes run according to existing Round Robin then it gives AWT as 23 ms, so enhanced Round Robin improves the waiting time.

3.3 Performance Evaluation

Performance of multiprogramming and multiuser operating system is largely dependent on their effectiveness in allocating system resources. An active process simultaneously requires following resources in order to execute [1, 3, 7]

Processors
Main Memory
I/O devices
Secondary storage

The inexact nature of practical schedulers make performance evaluation an important tool for assessing the effectiveness of existing systems and for estimating the behavior of the new systems as they are being designed. Performance evaluation tools of most interest to operating system designer are simulation and analytic models. Simulations are of particular interest during system design, when real hardware is not available for measurements and in situations where a reasonably accurate analytic model of the system is not mathematically tractable. There are different methods for CPU scheduling algorithm evaluation

(1) Deterministic evaluation
(2) Queuing Modeling
(3) Simulation
(4)Implementation

Deterministic modeling takes predetermined workload and defines performance of each algorithm. Different algorithms have different strengths, and no single one is ideal for every situation. Therefore, techniques are needed for analyzing the characteristics of each algorithm in order to make a suitable choice. Criteria for CPU scheduling includes such as

- Maximum CPU utilization under the constraints that the maximum response time is 1 second.
- Maximum throughput such that turnaround time is linearly proportional to execution time.
- Turnaround time is time taken by process to execute.
- Waiting Time is the sum of the periods spent waiting in the ready queue.

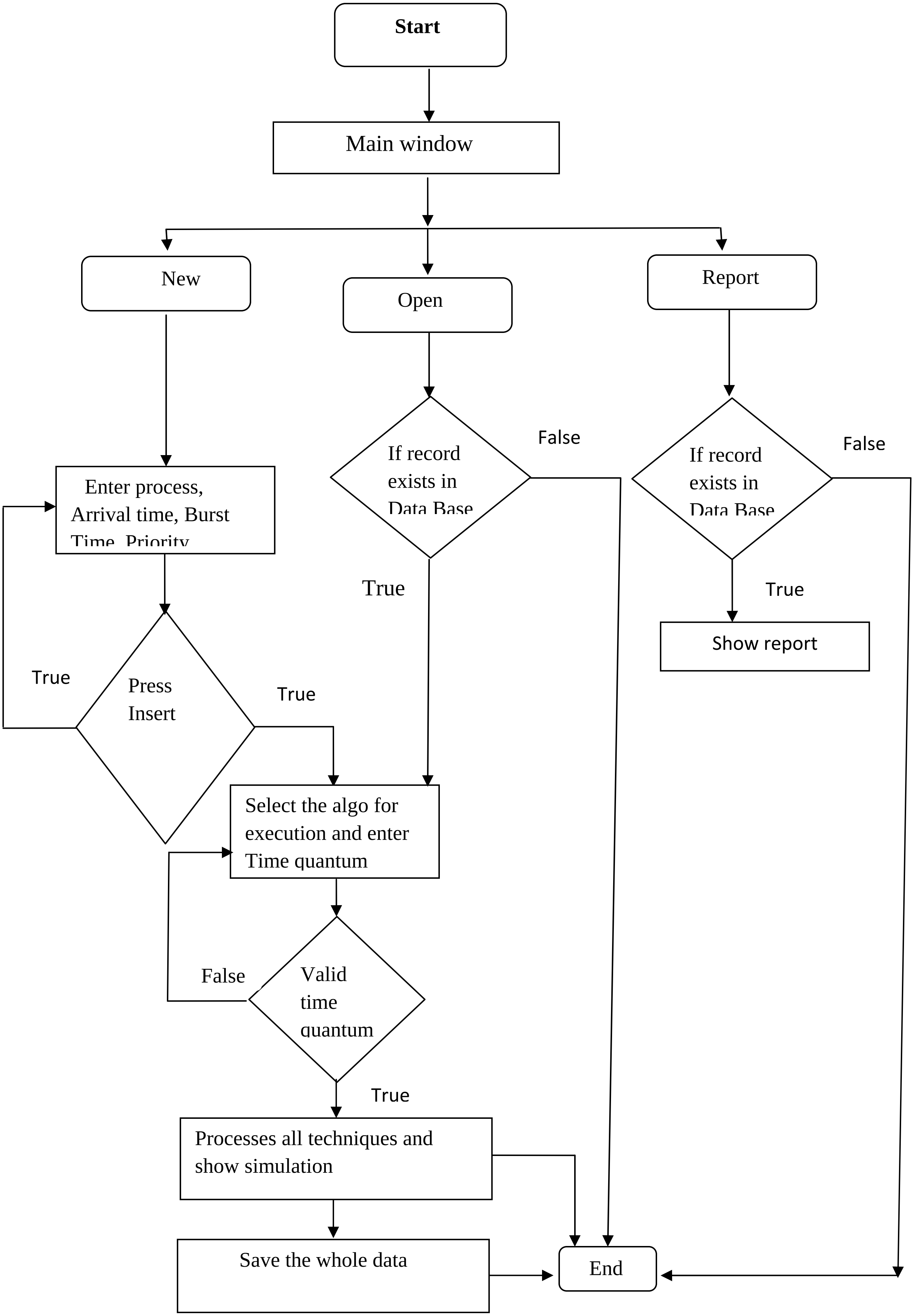
- Response Time is the interval between submissions of a request until the first response is produced.

Once the selection criterion is defined, various algorithms are evaluated under consideration. There are number of different evaluation method.

One of the evaluation method is called Analytic Evaluation. Analytic evaluation uses the given algorithm and the system workload to produce a formula or number that evaluates the performance of the algorithm for that workload. One type of analytic evaluation is deterministic modeling. This method takes a particular predetermined workload and defines the performance of each algorithm for that workload. In this work different CPU scheduling methodologies are evaluated using deterministic modeling and through simulation.

3.4 Introduction and design of developed Software

The simulator for CPU scheduling algorithms has been developed with a view to develop a software tool, which can be used for the study and evaluation of CPU scheduling algorithms. This software has been developed as a comprehensive software package, which runs a simulation, generates useful data to be used for performance evaluation of algorithm 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 a personal Computer as a windows application. It provides the user an opportunity to simulate a multiprogramming environment on a PC.The system is required to simulate creation, concurrent execution and termination of processes in a batch processing and multiprogramming environment. It should maintain system queues which include ready queues. It is required to record process state transitions and system data at run time. The system should use the data to compute algorithm evaluation parameters and to ascertain behavior of the modeled system. A user friendly and mouse driven graphical user interface (GUI) to be integrated in order to provide a user the opportunity to select an algorithm from the displayed menu. The system is required to simulate the execution of selected algorithm. The block Diagram of this system is given below

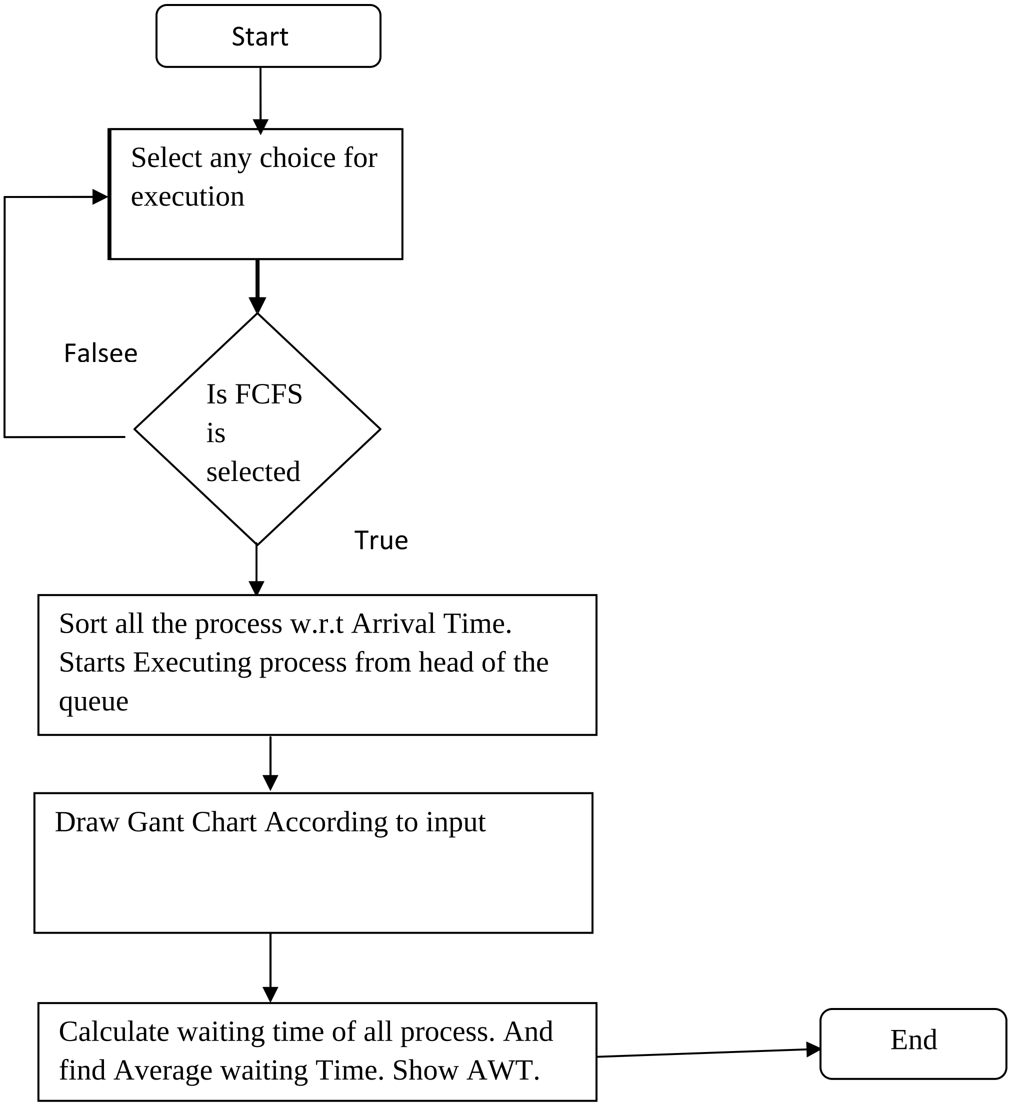

Start
Main window
New
Open
Report
If record exists in Data Base
False
If record exists in Data Base
False
Enter process, Arrival time, Burst Time, Priority
True
Show report
True
True
Press Insert
True
Select the algo for execution and enter Time quantum
False
Valid time quantum
True
Processes all techniques and show simulation
Save the whole data
End

Flow chart of Analyzer

The system designs of these scheduling algorithms are outlined below.

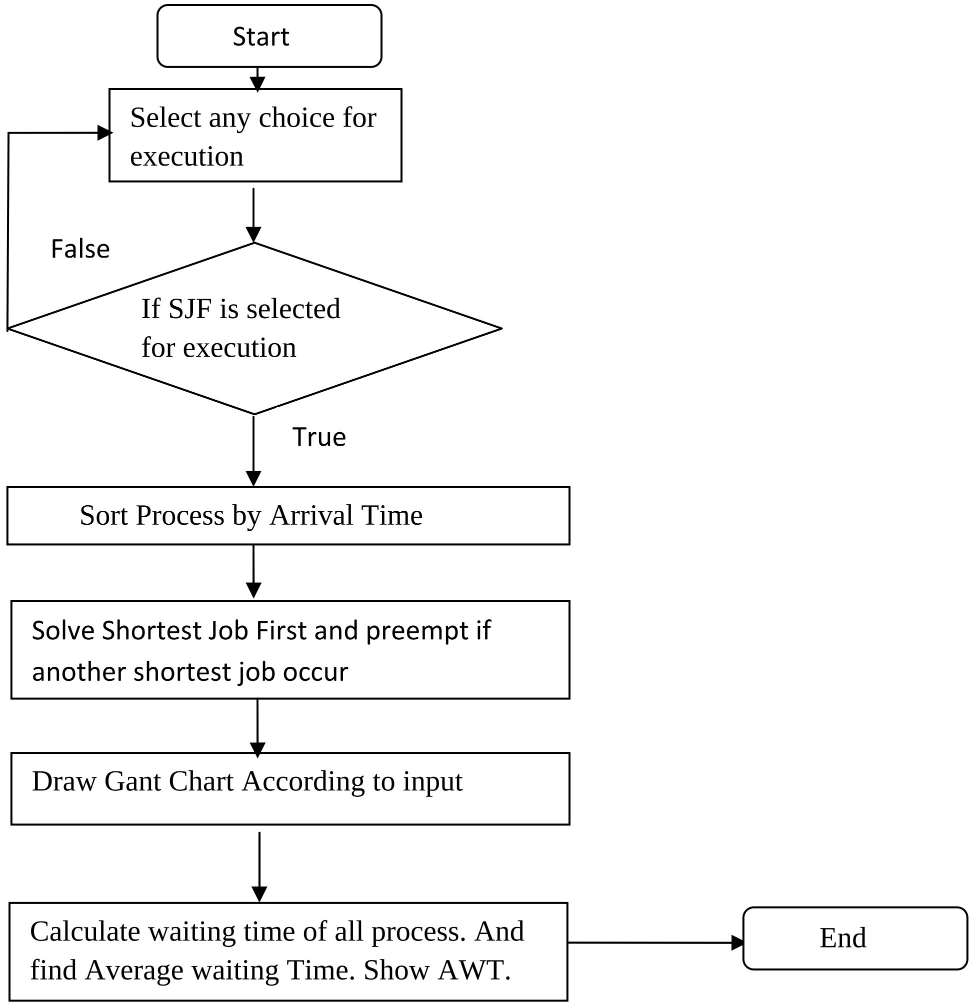
3.4.1 Design of FCFS Scheduling Algorithm

For this algorithm ready queue is maintained as FIFO queue.PCBs of the processes, submitted to the system is linked at the tail of the queue. The Algorithm dispatches processes from head of the ready queue and simulates its execution through CPU.The process being simulated is terminated on completing the task and deleted from the system. Next process is then dispatched from the head of the ready queue. During simulation, algorithm performance data is displayed on the screen. It also saves the process data and performance parameter in data base as well. The flow chart for FCFS is given below.


Start
Select any choice for execution
Falsee
Is FCFS is selected
True
Sort all the process w.r.t Arrival Time. Starts Executing process from head of the queue
Draw Gant Chart According to input
Calculate waiting time of all process. And find Average waiting Time. Show AWT.
End

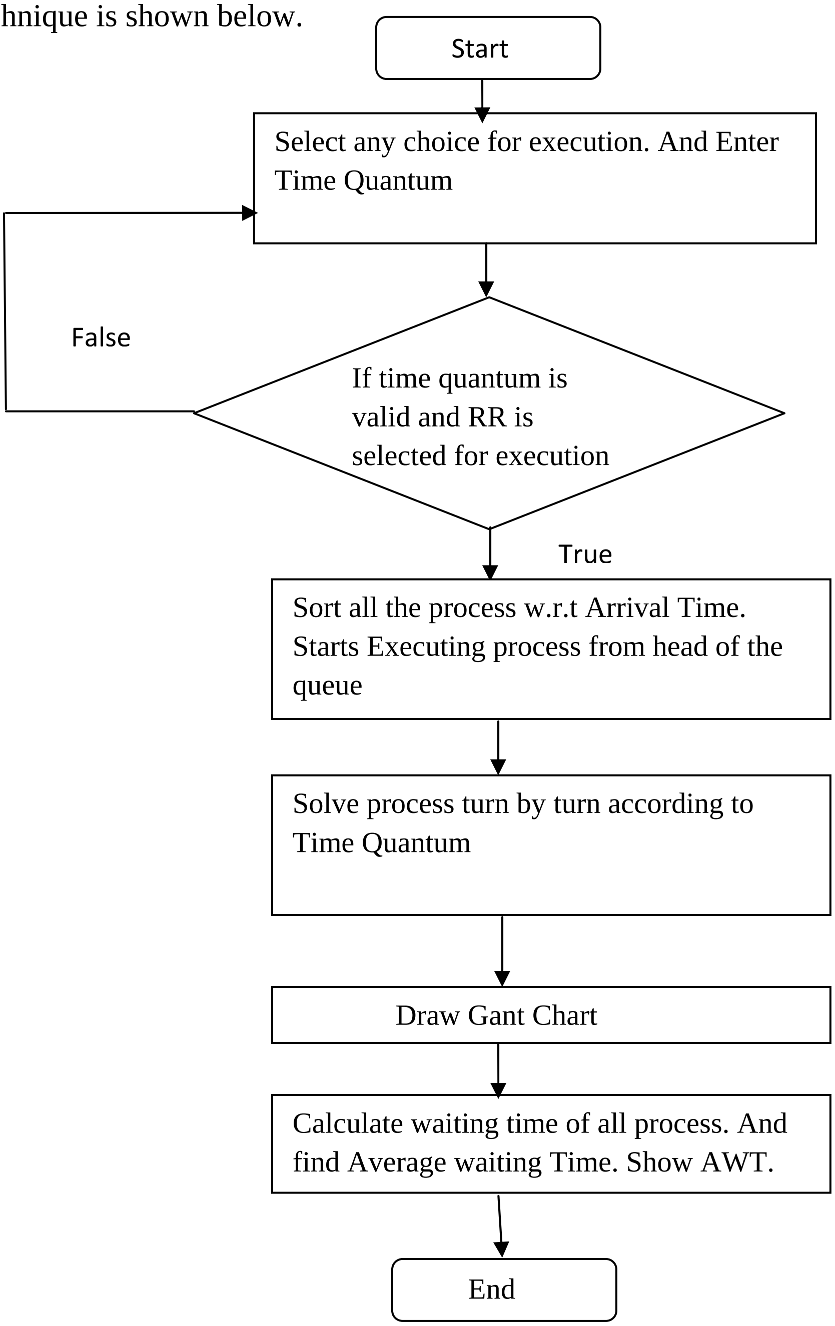
3.4.2 Design of SJF Scheduling Algorithm

PCBs are linked in ready queue in order of CPU bursts length, with the shortest burst length at the head of the queue. On arrival of new processes in the ready queue the system compares the CPU burst time of newly created PCB with the remaining time of PCB at execution. If newly created PCB has less CPU burst time than PCB at running; then preemption occurs and turn is given to the new PCB for execution. During execution, performance data can be visualized empirically and graphically; and can be stored in data base for future use. Flow chart of implemented SJF is shown below


Start
Select any choice for execution
False
If SJF is selected for execution
True
Sort Process by Arrival Time
Solve Shortest Job First and preempt if another shortest job occur
Draw Gant Chart According to input
Calculate waiting time of all process. And find Average waiting Time. Show AWT.
End

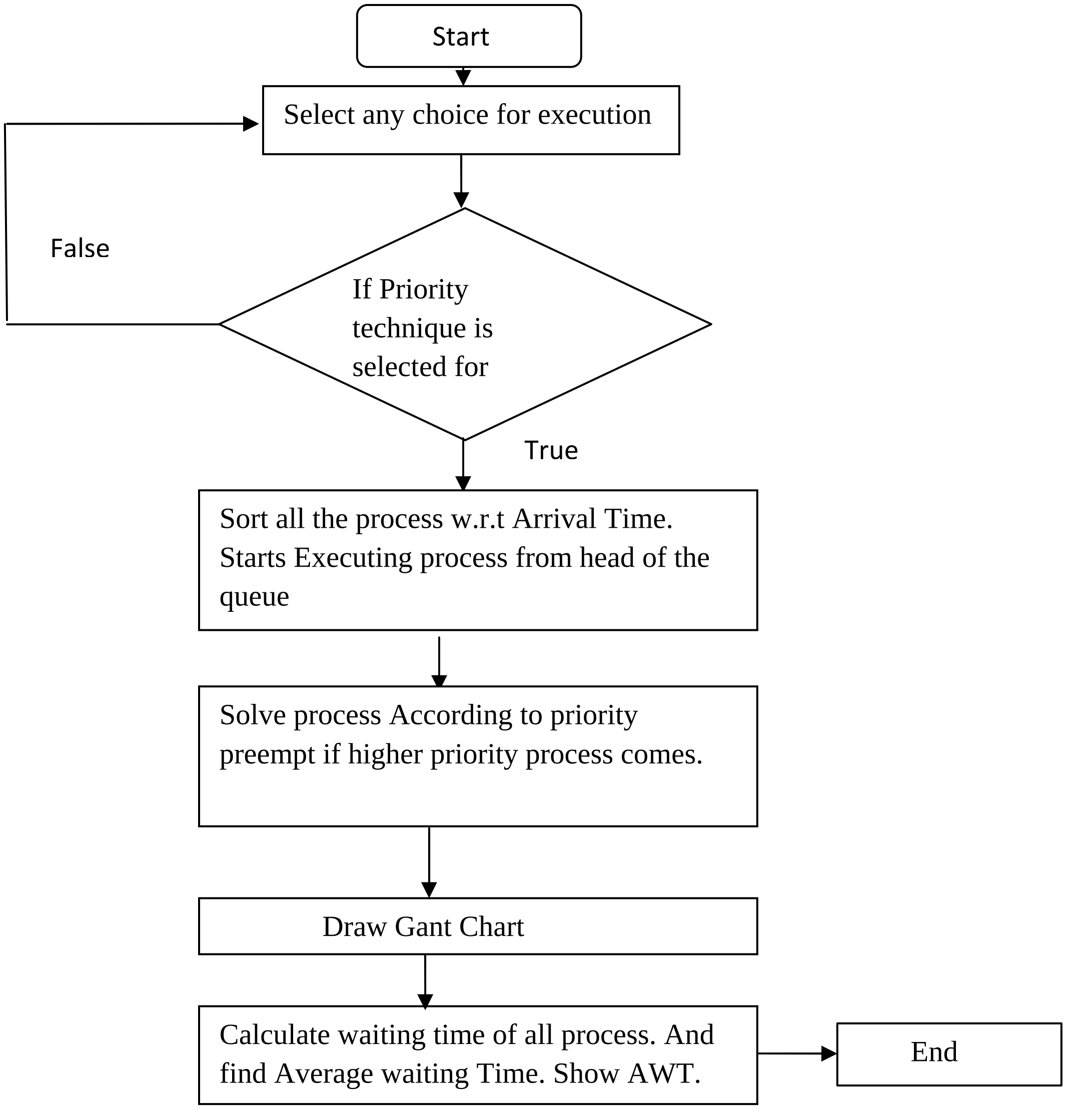
3.4.3 Design of Round Robin scheduling algorithm

Ready queue is maintained as FIFO queue. Processes are selected from the head of the ready queue. Processes being executed is preempted in expiry of time quantum, which is a user defined variable. A preempted process's PCB is linked at the tail of the ready queue.PCB completing CPU burst before the expiry of time quanta is terminated and removed from the queue. During execution, performance data can be visualized as well as can be stored in data base. Flow chart for this technique is shown below.


Start
Select any choice for execution. And Enter Time Quantum
False
If time quantum is valid and RR is selected for execution
True
Sort all the process w.r.t Arrival Time. Starts Executing process from head of the queue
Solve process turn by turn according to Time Quantum
Draw Gant Chart
Calculate waiting time of all process. And find Average waiting Time. Show AWT.
End

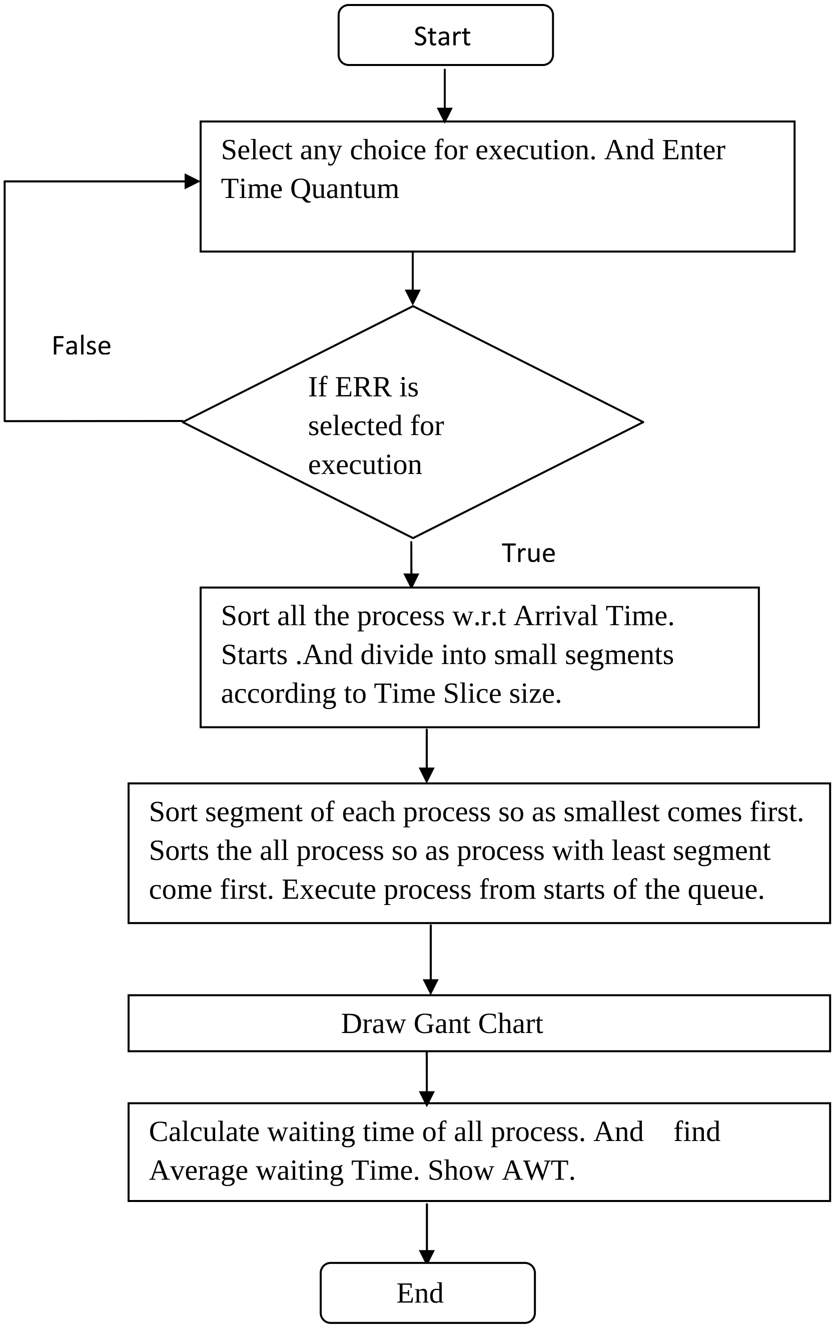
3.4.4 Design of Priority scheduling Algorithm

PCBs are linked in the ready queue in the order of user defined priorities with PCB having highest priority (lowest number) 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 is terminated and removed from the system. In preemptive priority Scheduling algorithm preemption occurs if the newly created PCB has higher priority than the PCB in execution. During execution, performance data can be visualized as well as can be stored in a data base.


Start
Select any choice for execution
False
If Priority technique is selected for
True
Sort all the process w.r.t Arrival Time. Starts Executing process from head of the queue
Solve process According to priority preempt if higher priority process comes.
Draw Gant Chart
Calculate waiting time of all process. And find Average waiting Time. Show AWT.
End

3.4.5 Design of Optimized Round Robin Scheduling.

Ready queue is maintained as a FIFO queue. Then processes are divided into segments according to size of time quantum given by user. Then ready queue is again sorted according to segments. The process which have least segment/ is placed at head of the queue. These segments are being executed and preempted on complete execution of segment. During execution, performance data can be visualized as well as stored in data base. Flow diagram of its design implementation is given below.

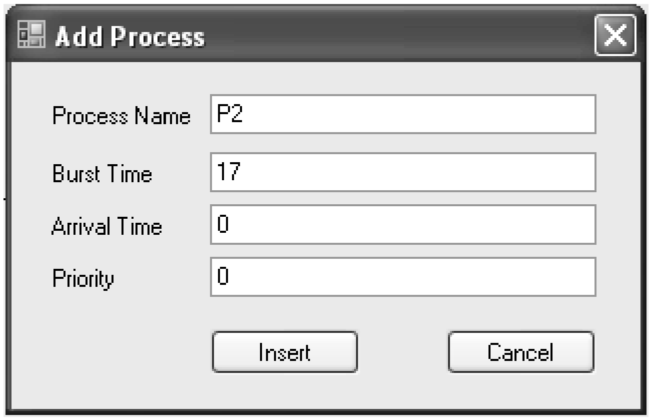

Start
Select any choice for execution. And Enter Time Quantum
False
If ERR is selected for execution
True
Sort all the process w.r.t Arrival Time. Starts .And divide into small segments according to Time Slice size.
Sort segment of each process so as smallest comes first. Sorts the all process so as process with least segment come first. Execute process from starts of the queue.
Draw Gant Chart
Calculate waiting time of all process. And find Average waiting Time. Show AWT.
End

3.5 Input of System

The system displays options for taking input data for scheduling algorithm. It takes complete process information

- Process Id
- CPU Burst Time
- Arrival Time
- Priority
- Time Quantum

The system also ensures the correctness of data by imposing checks on input values. The value of the CPU Burst time, arrival time, and time quantum should be some positive values. The value of priority should be between 0 and 9;) is taken as the highest priority while 9 is the lowest priority. The system simulates concurrent execution of processes. There is also no limit on the number of process in a batch. A user can program the system to simulate concurrent execution of a batch consisting of thousands of processes. The data entered by user gets saved into data base.. Then user can select scheduling algorithm from given option and run simulation. A user can program the system to simulate concurrent execution of a batch consisting of thousands of processes. A user can also simulate more than one algorithm for same set of processes to make comparative analysis. A user can visualize quantative as well as graphical results in the form of Gant chart. Figure 3.1 shows how data for process can be entered into the system.



Process Name: P2

Burst Time: 17

Arrival Time: 0

Priority: 0

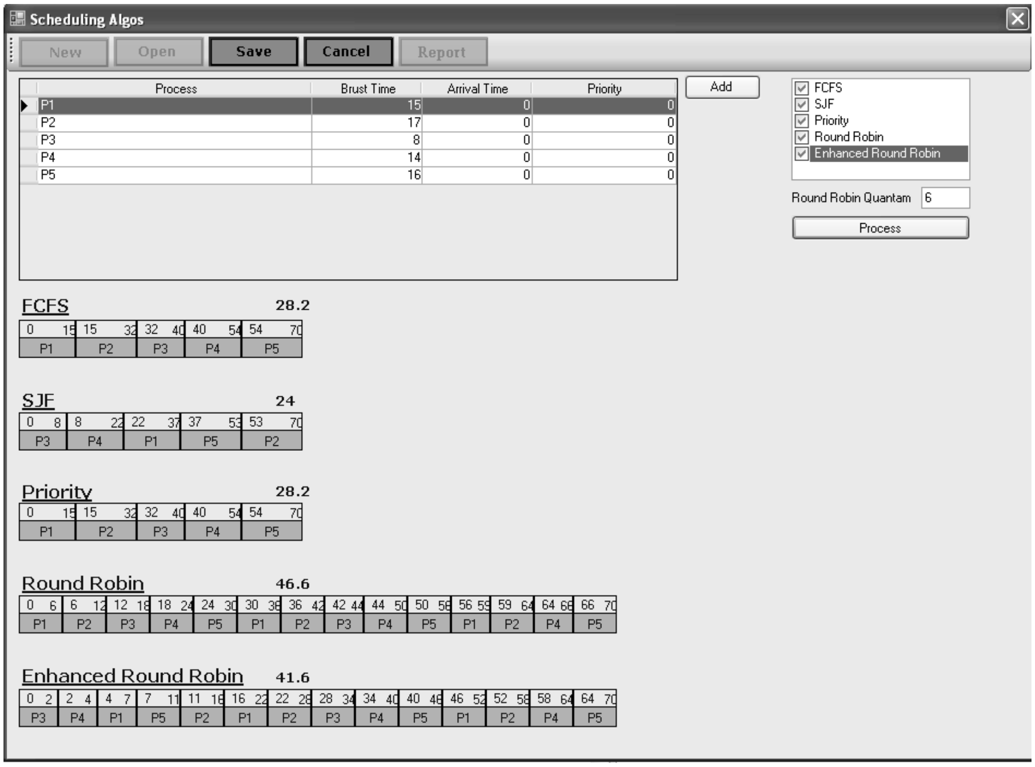
Figure 3.1 Input window

3.6 Output of System

The system stores simulation data at run time in data base. The system provides the option of saving simulation in database. A user can store complete process history as well as performance factor in data base .The data is essential for:

- The performance evaluation of algorithms
- Detailed analysis of algorithms for the purpose of research
- User can run any saved simulation any time.

System also shows result of execution in form of Gantt chart and also can save this data into data base, as shown in Figure 3.2 and Figure 3.3.



Process	Brust Time	Arrival Time	Priority
P1	15	0	0
P2	17	0	0
P3	8	0	0
P4	14	0	0
P5	16	0	0

☑ FCFS
☑ SJF
☑ Priority
☑ Round Robin
☑ Enhanced Round Robin

Round Robin Quantam 6

FCFS 28.2

0 15	15 32	32 40	40 54	54 70
P1	P2	P3	P4	P5

SJF 24

0 8	8 22	22 37	37 53	53 70
P3	P4	P1	P5	P2

Priority 28.2

0 15	15 32	32 40	40 54	54 70
P1	P2	P3	P4	P5

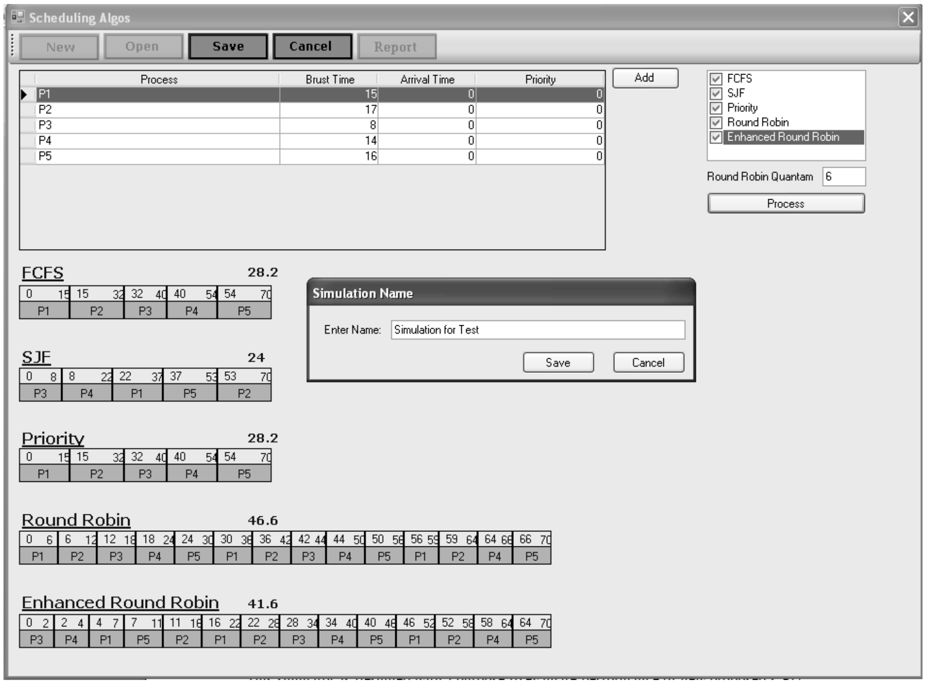
Round Robin 46.6

0 6	6 12	12 18	18 24	24 30	30 36	36 42	42 44	44 50	50 56	56 59	59 64	64 66	66 70
P1	P2	P3	P4	P5	P1	P2	P3	P4	P5	P1	P2	P4	P5

Enhanced Round Robin 41.6

0 2	2 4	4 7	7 11	11 16	16 22	22 28	28 34	34 40	40 46	46 52	52 58	58 64	64 70
P3	P4	P1	P5	P2	P1	P2	P3	P4	P5	P1	P2	P4	P5

Figure 3.2 Output window



Process	Brust Time	Arrival Time	Priority
P1	15	0	0
P2	17	0	0
P3	8	0	0
P4	14	0	0
P5	16	0	0

☑ FCFS
☑ SJF
☑ Priority
☑ Round Robin
☑ Enhanced Round Robin

Round Robin Quantam: 6

Enter Name: Simulation for Test

FCFS 28.2

0 15	15 32	32 40	40 54	54 70
P1	P2	P3	P4	P5

SJF 24

0 8	8 22	22 37	37 53	53 70
P3	P4	P1	P5	P2

Priority 28.2

0 15	15 32	32 40	40 54	54 70
P1	P2	P3	P4	P5

Round Robin 46.6

0 6	6 12	12 18	18 24	24 30	30 36	36 42	42 44	44 50	50 56	56 59	59 64	64 66	66 70
P1	P2	P3	P4	P5	P1	P2	P3	P4	P5	P1	P2	P4	P5

Enhanced Round Robin 41.6

0 2	2 4	4 7	7 11	11 16	16 22	22 28	28 34	34 40	40 46	46 52	52 58	58 64	64 70
P3	P4	P1	P5	P2	P1	P2	P3	P4	P5	P1	P2	P4	P5

Figure 3.3 Save window

3.7 Design of Data Structures

3.7.1 Processes

In a real multiprogramming system, once a program is submitted to the system, a process is created, that is, memory allocated and program loaded into the memory. Moreover, the operating system creates a PCB for the process.

This simulator is designed with a purpose to evaluate performance of new proposed CPU scheduling algorithm and compare with the performance of existing Round Robin and all other existing algorithm. Hence creation of a process are not required to perform actual computing work .Hence creation of a process does not require a real program to be leaded into the memory. Whereas the creation is accomplished by merely creating a PCB and allocating memory to it. Consequently, a process in this simulation is represented by PCB only. As actual computing work is not required to be done, PCBs are not required to store run time description of the processes, the space, however, is used for storing execution data of the simulation itself.

3.7.2 Design of Process Control Blocks (PCBs)

PCB consists of a number of data items grouped together. The data items, which can be of different data types, are known as members of the structures. Each data item can be identified by its own identifier [4,5,6].This system creates a PCB data structure consisting of following data members. Detail is given as below.

Members	**Data Type**	**Utilization**
Process Name	Varchar(50)	Name of process a unique identifier of the process
Burst Time	Bigint	Entered by user at run time. Actual amount of execution time.
Arrival Time	Bigint	Entered by user at run time
Priorty	Bigint	Run by user at run time .Will be used in priority Scheduling.
SimulationId	Bigint	It will be used for Simulation
ProcessColor	Varchar(50)	It will be given by system to distinguish different processes

Table 3.1 Process Data

When a process is created by the system; then its relevant information are saved in data base. When process executes it pick this data and run accordingly. In the case of priority scheduling, system also encounters the priority of a process during its execution. In case of Round Robin and Optimized Round Robin system executes processes on the basis of time quantum.

3.7.3 Design of Simulation Data

System also stores information of different simulation of different set of process. Following information are saved for simulation in data base.

Members	Data Type	Utilization
SimulationId	Bigint	It is Unique Identifier for simulation.
SimulationName	Varchar(500)	It is used for Simulation. Given by user at run time.
FCFS	bit	It will check if user has selected Option for FCFS to simulate
FCFSAWT	Varchar(50)	It will store average waiting time for FCFS calculated by system

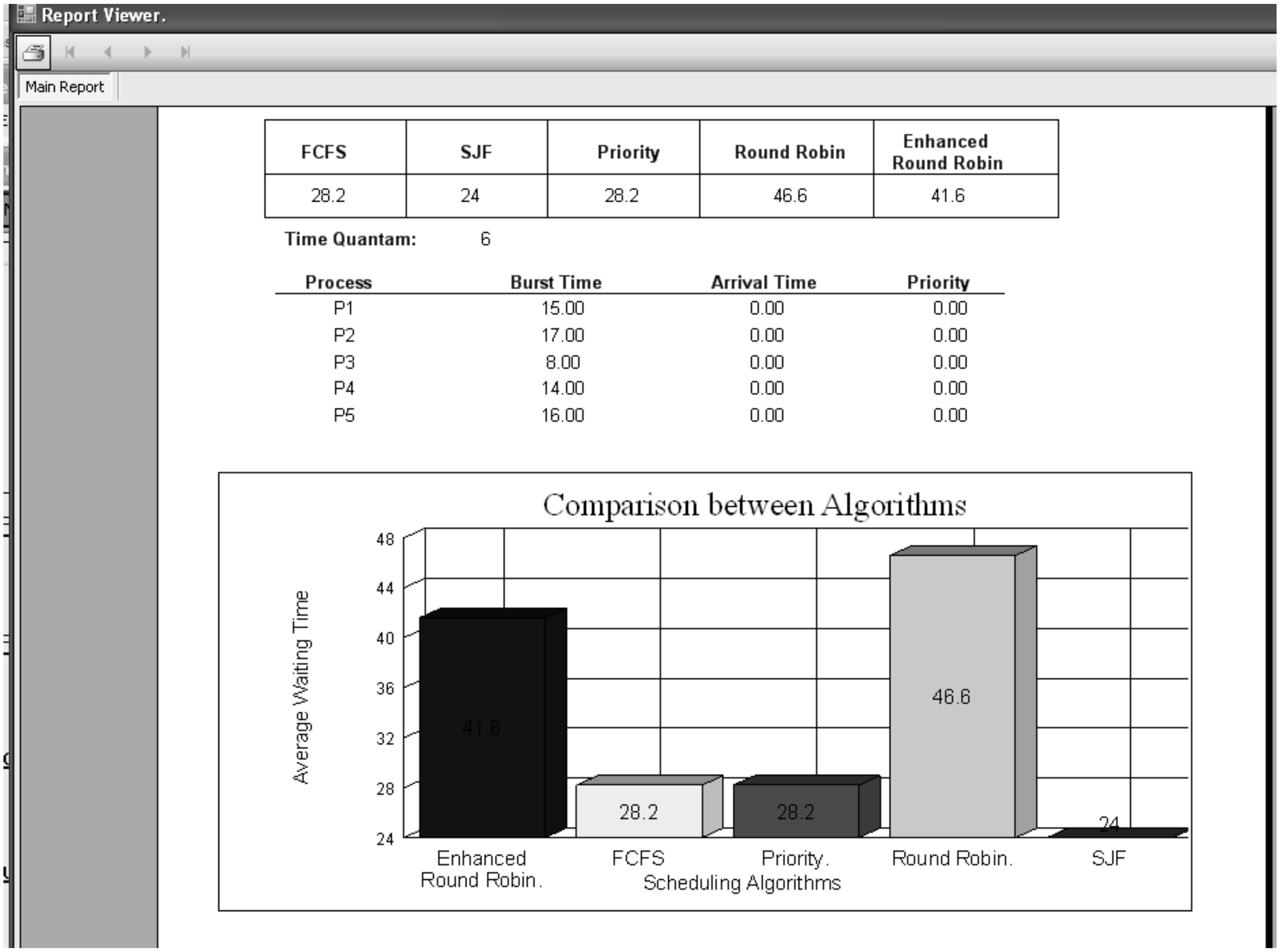
SJF	Bit	It will check if user has selected Option for SJF to simulate
SJFAWT	Varchar(50)	It will store average waiting time for SJF calculated by system
RR	Bit	It will check if user has selected Option for RR to simulate
RRAWT	Varchar(50)	It will store average waiting time for Round Robin calculated by system
ERR	Bit	It will check if user has selected Option for Enhanced Round Robin to simulate
ERRAWT	Varchar(50)	It will store average waiting time for Enhanced Round Robin calculated by system.
RoundRobinQuantam	Varchar(50)	It will store time slice for Round Robin and Enhanced round robin given by user.

Table 3.2 Data Related to Simulation

3.7.4 Graphical User Interface

A user friendly and mouse driven GUI has been developed. It provides a menu driven interactive environment. User can also open the report of any one of saved simulation and view its result in quantative analysis as well as in graphical format. As this system saves the all out put related to scheduling algorithm in to data base, so this system also shows reports in the form of performance comparison between scheduling algorithm as shown in Figure 3.4

This reports shows AWT time of all algorithm which have been run in one simulation. it also shows comparison between all algorithm.



FCFS	SJF	Priority	Round Robin	Enhanced Round Robin
28.2	24	28.2	46.6	41.6

Time Quantam: 6

Process	Burst Time	Arrival Time	Priority
P1	15.00	0.00	0.00
P2	17.00	0.00	0.00
P3	8.00	0.00	0.00
P4	14.00	0.00	0.00
P5	16.00	0.00	0.00

Figure 3.4 Report Viewer

CHAPTER4

Results

And

Discussion

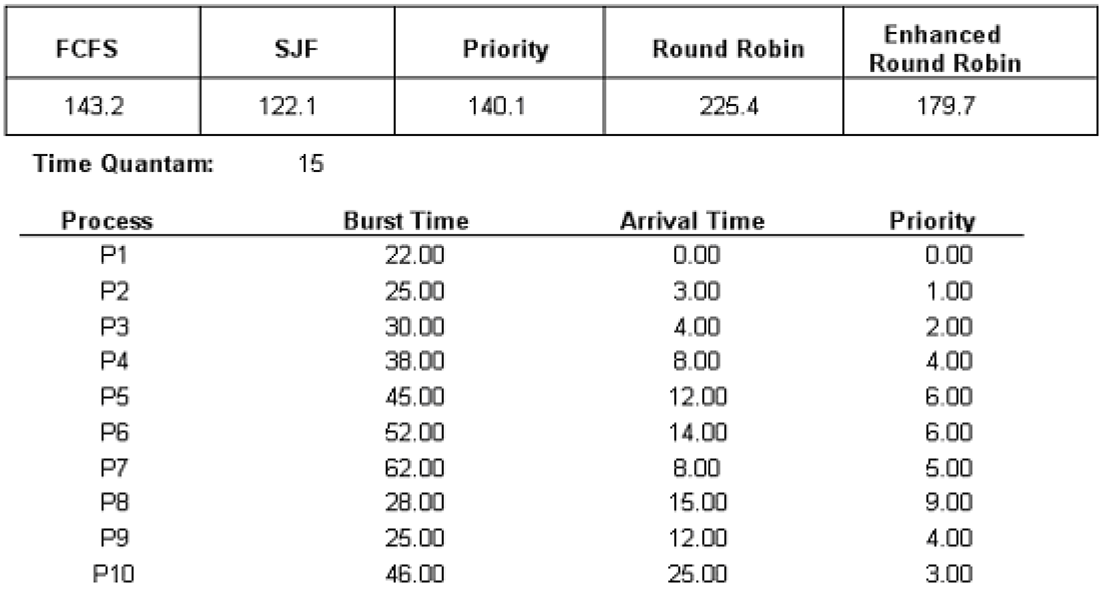
FCFS, SJF, Round-Robin, Priority and Optimized Round Robin techniques are implemented in the scheduler analyzer. This work involves development of a simulator for CPU scheduling with view to optimization. It has been developed for deterministic evaluation of CPU algorithm. And on the basis of evaluation an enhanced Round Robin has been made and implemented, which shows better result as compared to existing Round Robin Scheduling for time sharing systems. .Following analysis provides detailed comparison of all above techniques.

4.1 Comparative Analysis of all Algorithms.

Input of ten processes is given to scheduler analyzer and time quantum is taken as such that most of process completes their execution in initial two or three turns.

Process Set #1

Figure 4.1 and 4.2 show that when average waiting time of SJF is very small as compared to other methods. It also shows that Optimized Round Robin has less average waiting time as compared to existing Round Robin. So in this case, performance of Optimized Round Robin is better than existing Round Robin.



FCFS	SJF	Priority	Round Robin	Enhanced Round Robin
143.2	122.1	140.1	225.4	179.7

Time Quantam: 15

Process	Burst Time	Arrival Time	Priority
P1	22.00	0.00	0.00
P2	25.00	3.00	1.00
P3	30.00	4.00	2.00
P4	38.00	8.00	4.00
P5	45.00	12.00	6.00
P6	52.00	14.00	6.00
P7	62.00	8.00	5.00
P8	28.00	15.00	9.00
P9	25.00	12.00	4.00
P10	46.00	25.00	3.00

Figure 4.1 Input for process set #1

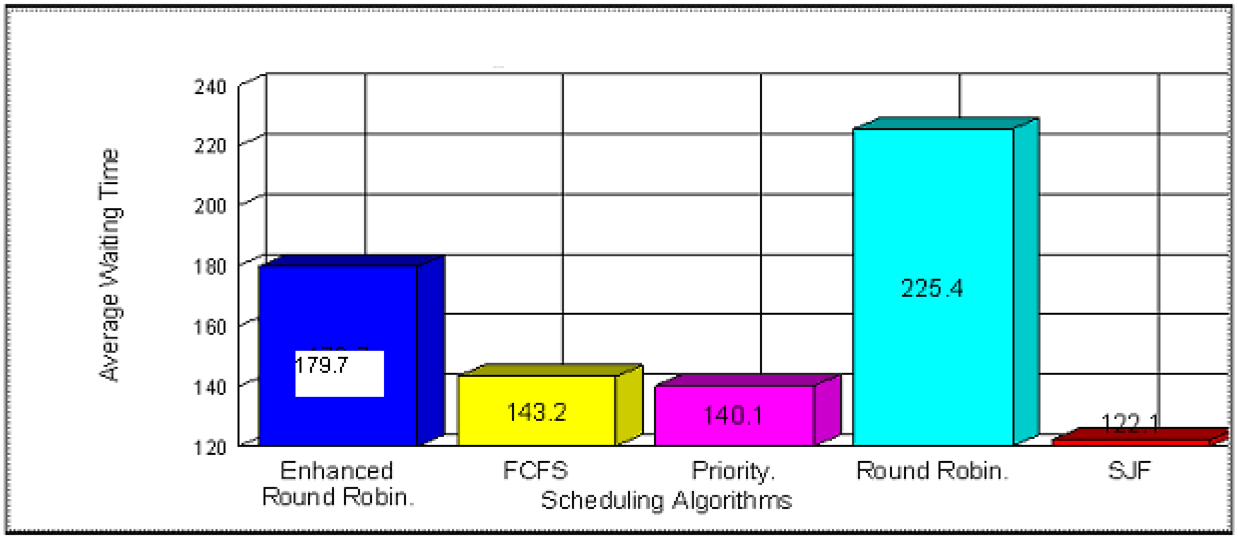
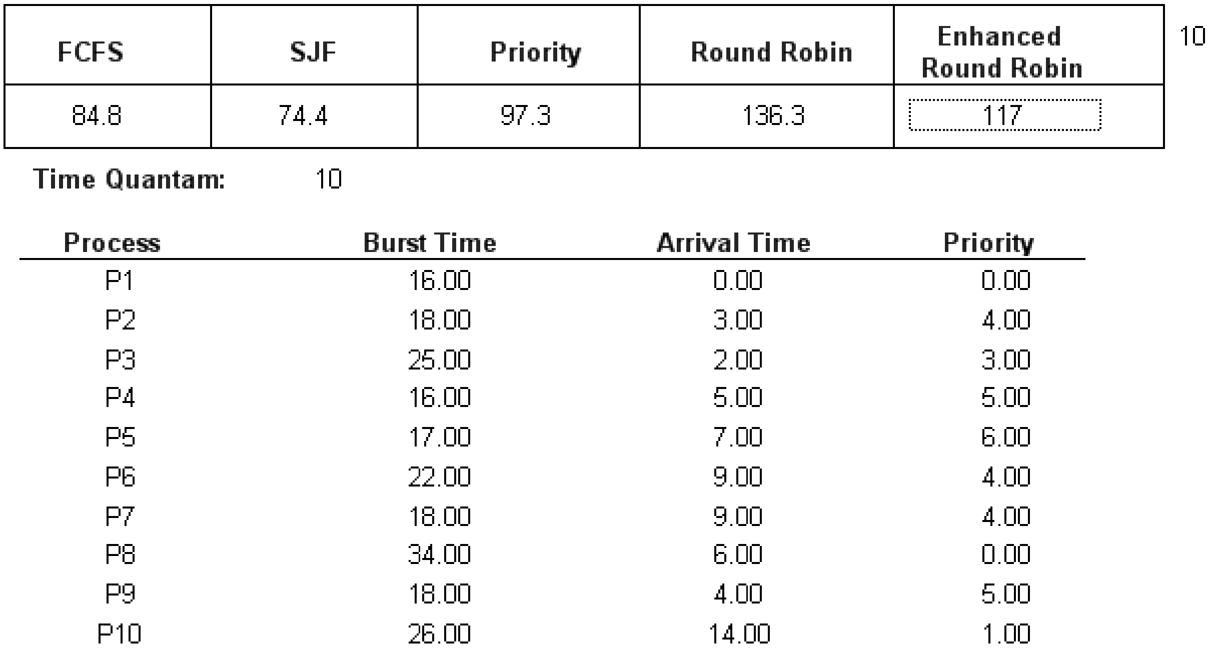


Figure 4.2: Comparison between all algorithms for processes set# 1

Figure

Process Set # 2



FCFS	SJF	Priority	Round Robin	Enhanced Round Robin
84.8	74.4	97.3	136.3	117

10

Time Quantam: 10

Process	Burst Time	Arrival Time	Priority
P1	16.00	0.00	0.00
P2	18.00	3.00	4.00
P3	25.00	2.00	3.00
P4	16.00	5.00	5.00
P5	17.00	7.00	6.00
P6	22.00	9.00	4.00
P7	18.00	9.00	4.00
P8	34.00	6.00	0.00
P9	18.00	4.00	5.00
P10	26.00	14.00	1.00

Figure 4.3 Input for process set #2

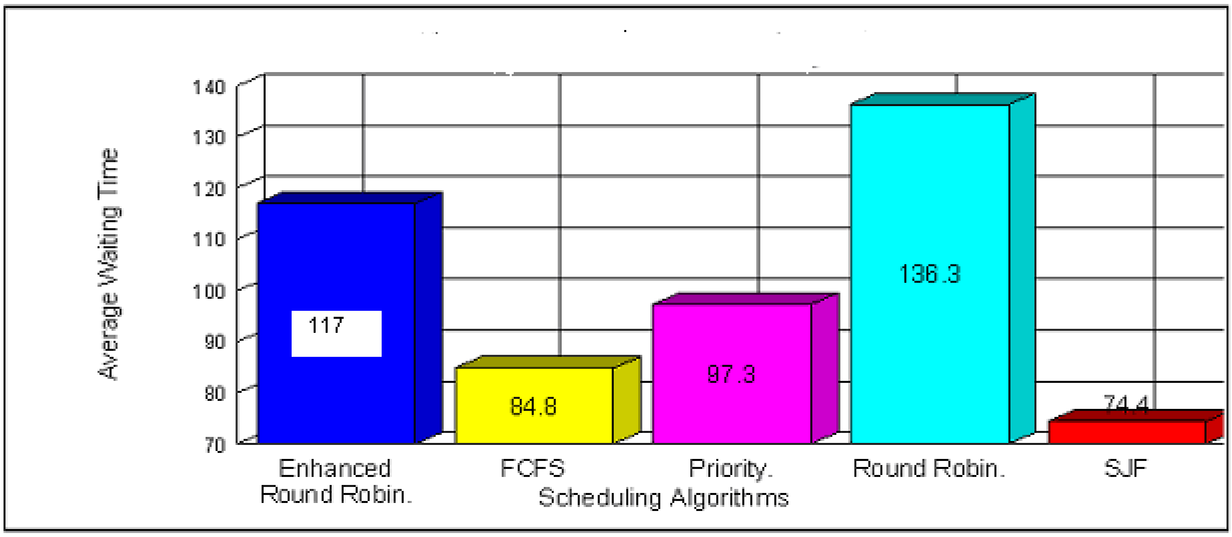
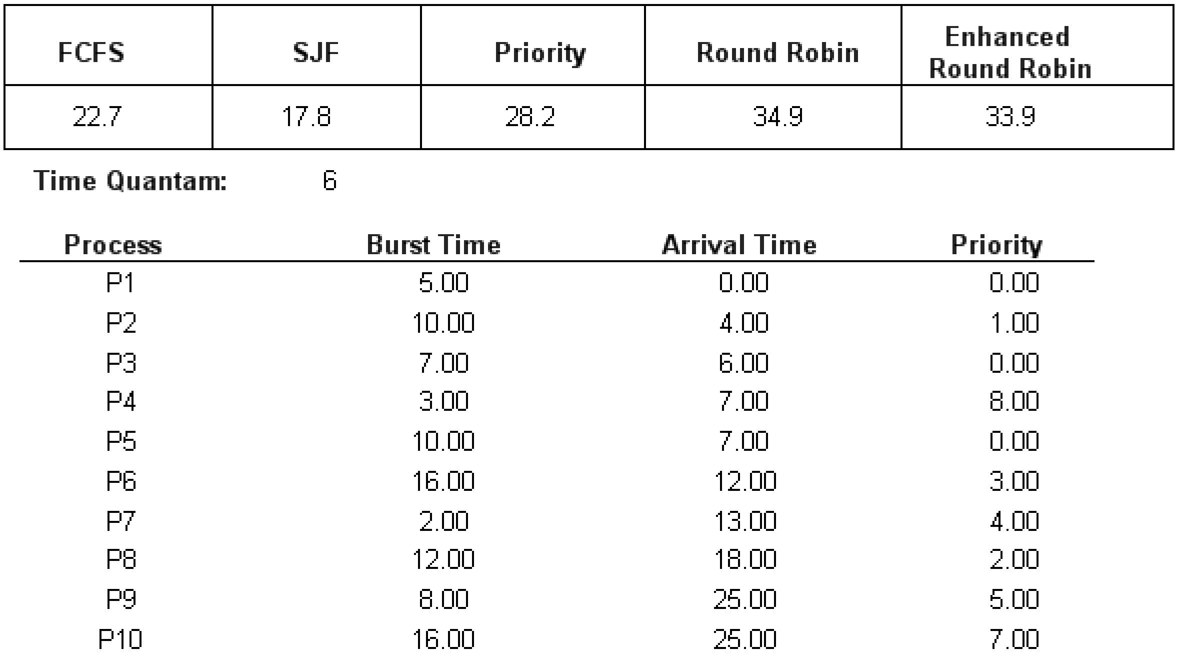


Figure 4.4 Comparison between all algorithms for processes set# 2

Figure 4.3 and 4.4 show that SJF is better as compared to other methods, and performance of FCFS is better than priority algorithm. Similarly the performance of Optimized Round Robin is also better than existing Round Robin.

Process Set #3



FCFS	SJF	Priority	Round Robin	Enhanced Round Robin
22.7	17.8	28.2	34.9	33.9

Time Quantam: 6

Process	Burst Time	Arrival Time	Priority
P1	5.00	0.00	0.00
P2	10.00	4.00	1.00
P3	7.00	6.00	0.00
P4	3.00	7.00	8.00
P5	10.00	7.00	0.00
P6	16.00	12.00	3.00
P7	2.00	13.00	4.00
P8	12.00	18.00	2.00
P9	8.00	25.00	5.00
P10	16.00	25.00	7.00

Figure 4.5 Input for process set #3

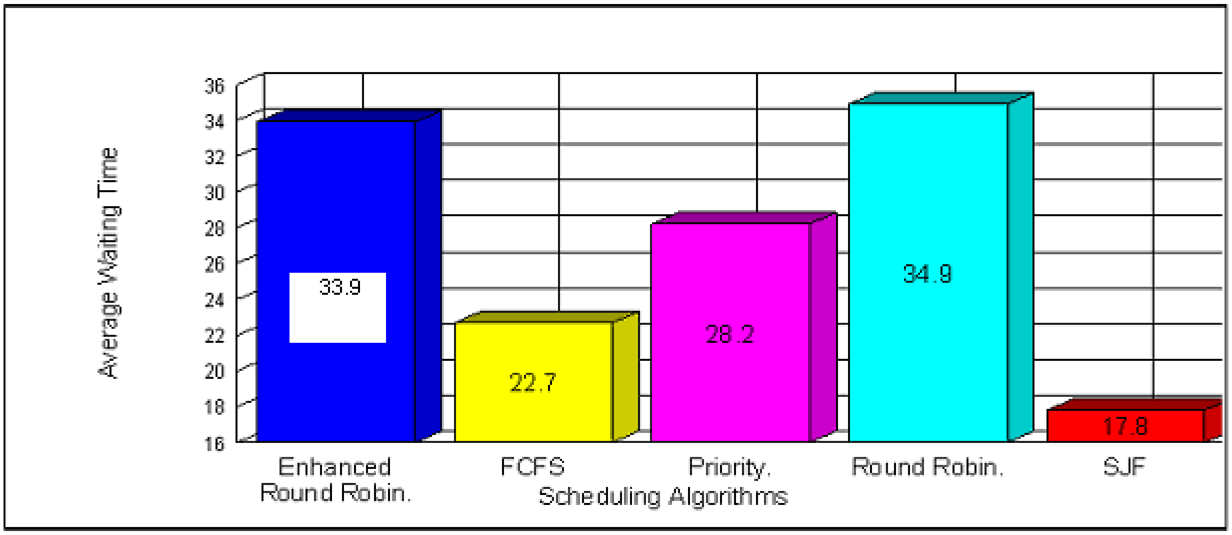
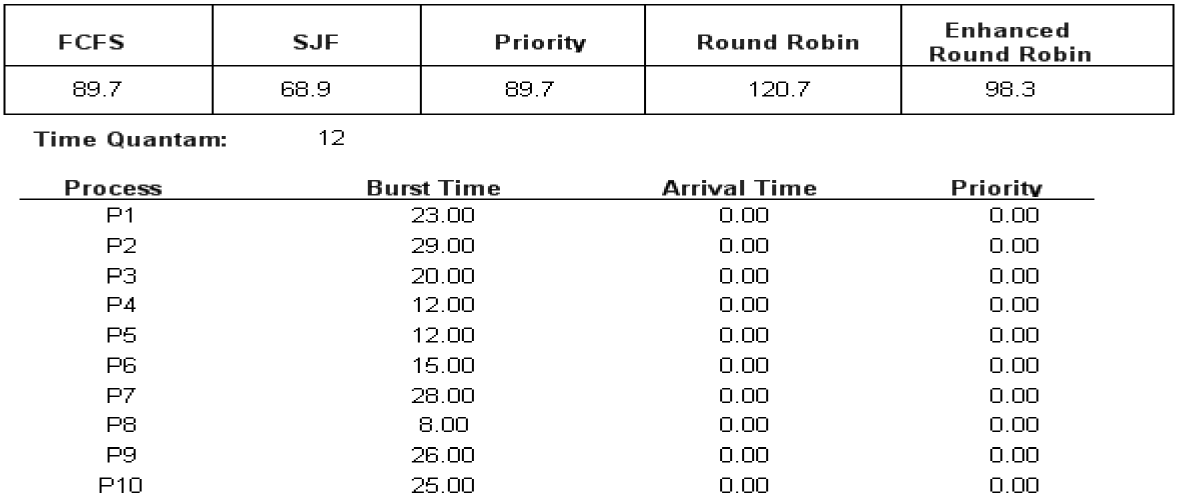


Figure 4.6 Comparison between all algorithms for processes set# 3

Figure 4.5 and 4.6 show that SJF is better as compared to other methods, and performance of FCFS is better than priority algorithm. Similarly the performance of Optimized Round Robin is also better than existing Round Robin.

Process Set #4

Now if ten processes are given to system as input such that, Arrival time and priority are taken as 0 then following results can be found as shown in Figure 4.7 and 4.8.In this case again SJF shows better performance, But AWT of both FCFS and priority algorithms is same. Optimized Round Robin is giving better performance than existing Round Robin.



FCFS	SJF	Priority	Round Robin	Enhanced Round Robin
89.7	68.9	89.7	120.7	98.3

Time Quantam: 12

Process	Burst Time	Arrival Time	Priority
P1	23.00	0.00	0.00
P2	29.00	0.00	0.00
P3	20.00	0.00	0.00
P4	12.00	0.00	0.00
P5	12.00	0.00	0.00
P6	15.00	0.00	0.00
P7	28.00	0.00	0.00
P8	8.00	0.00	0.00
P9	26.00	0.00	0.00
P10	25.00	0.00	0.00

Figure 4.7 Input for process set # 4

Graphical representation of above data is shown as follows.

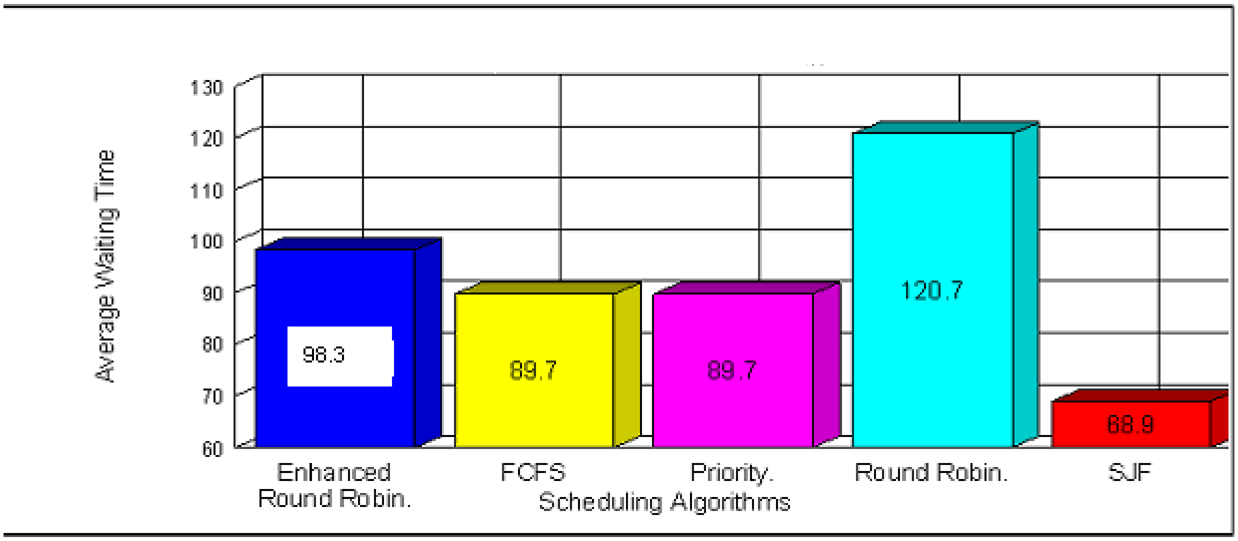


Figure 4.8 Comparison between all algorithms for processes set# 4

4.2 Summary

In FCFS priority is given to the process which comes first. Assume a set of two processes in which first process has a larger burst time and second process has a smaller burst time. The CPU will start executing the process which arrived first. This larger CPU burst process will start consuming time of the CPU. As a result, second process with smaller CPU burst will keep waiting due to starvation. In other words, FCFS has a disadvantage of starvation in certain cases. Second disadvantage of FCFS scheduling is that it is non- preemptive. Once CPU is allocated to a process, that process will keep hold of it till completion. Third disadvantage is that the FCFS algorithm is particularly troublesome for time sharing systems, where each user needs to get a share of the CPU after a regular interval. It is devastating to allow one process to keep the CPU for an extended period of time. As a result FCFS scheme is not useful in scheduling interactive users, because it cannot guarantee good response time.

Priority scheduling algorithm preserves all of the drawbacks of FCFS. A major problem with priority scheduling algorithm is indefinite blocking (or starvation). Process that is ready to run but lacking the CPU can be considered blocked. Priority scheduling algorithm leaves some low priority processes waiting indefinitely for the CPU. In a heavily loaded computer system, a steady stream of higher priority processes prevents a low priority process from getting the CPU for ever.

The priority scheduling with preemption has higher overhead than non-preemptive priority scheduling. It must keep track of the priority of the running job, and must handle occasional preemption. Arriving process with higher priority would be given CPU almost immediately. Low priority jobs, however, have an even longer mean waiting time.

Because of the drawbacks discussed earlier, we are left only with SJF, Round Robin, and Enhanced Round Robin.

Table 4.1 compares Average Waiting Time (AWT) for SJF, Round Robin and Enhanced Round Robin. Each record is based on a different numbers of processes, so to have a more solid comparison. It shows that as the numbers of processes increase, Average Waiting Time (AWT) of three algorithms also increases, but with equal proportions.

No of Processes	Scheduling Algorithms		
	AWTfor SJF(ms)	AWT for Round Robin(ms)	AWT for Enhanced Round Robin(ms)
25	1001.96	2016.6	1543.28
50	1600.1	3287.42	2461.46
75	2285.88	4560.84	3384.51
100	3105.54	6058.3	4649.32
125	4719.41	9108.89	6756.7
150	4618.68	8565.85	6511.36
175	5534.68	11158.9	7955.34
200	6741.86	13229.01	9703.33
225	7531.56	14523.01	10859.25
250	8396.1	16376.97	12307.39
275	9051.89	18097.39	13061.08
300	10100.76	19326.91	14406.23

Table 4.1 AWT of Scheduling algorithms for different number of processes

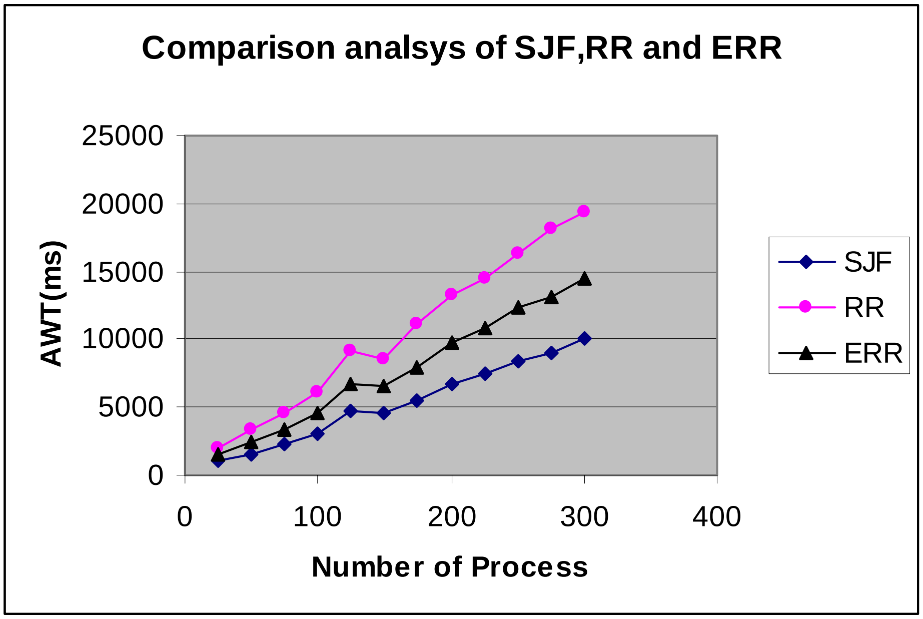


Figure 4.9 AWT of scheduling algorithms for different number of processes in graph

Figure 4.9 shows graphical representation of input data given in Table 4.5. SJF shows best results in all cases. When there are 100 processes, then AWT for SJF is 3.10554 seconds, AWT for Round Robin is 6.05 seconds and AWT for Enhanced Round Robin is 4.65 seconds. Similar results are also observed for a set of 175 processes where AWT for SJF is 5.53 seconds, for Round Robin it is 11.1 seconds and for Enhanced Round Robin it is 7.95 seconds.Similary if number of processes are increased to 200, the AWT for SJF is 6.74 seconds, RR is 13.22 seconds and ERR is 9.7seconds.When number of processes are 300 then AWT for SJF is 10.10 seconds, RR is 19.32 seconds and ERR it is 14.14 seconds. It describes that AWT of Enhanced Round Robin is much better than Round Robin.

SJF shows least AWT in all cases. So, it is the most efficient technique among all others. But it is not possible to find the next CPU burst. So SJF is difficult to implement.

It is only the Round Robin algorithm which gives good performance for time sharing systems. It prevents all processes from starvation. But it has more Average Waiting Time (AWT) as compared to SJF. There is a need to minimize AWT of Round Robin. Enhanced Round Robin has been designed and implemented for this purpose. All the tabulated results clearly indicate that it provides lesser AWT in all cases. It’s major reason is that processes waiting in the ready queue are re-arranged according to time quantum (i.e. Processes with smaller segment will be given CPU first).This modification in the RR technique works well and gives enhanced performance for time sharing systems.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

Computer systems supporting multiprogramming execute multiple programs concurrently. One of the objective of multiprogramming is to maximize resource utilization that is achieved by sharing system resources among multiple user and system processes. Efficient resource sharing depends on efficient scheduling of competing processes.

Process scheduling is an important aspect of multiprogramming operating system. 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 and deterministing being the most accurate of all is commonly employed for system’s performance evaluation despite the fact that they require complex programming for developing an efficient analyzer/simulator.

This work involves development of a simulator for proposed as well as existing CPU scheduling algorithms with view to get optimized performance of CPU scheduling algorithm in time sharing systems. Using this simulator deterministic evaluation of CPU scheduling algorithm can also be performed. This simulator can be used for measuring performance of different scheduling algorithms; implementation of operating system and for the understanding and training of Students. It simulates the following algorithms

- First Come First Serve(FCFS) scheduling
- Shortest Job First (SJF) scheduling
- Priority Scheduling
- Round Robin Scheduling
- Enhanced Round Robin Scheduling

This work presents a simulator that uses graphical animation to convey the concepts of various scheduling algorithms for a single CPU. The simulator is unique in a number of respects. First, it uses a more realistic process model that can be configured easily by the user. Second, it graphically depicts each process in terms of what the process is currently doing against time. Using this representation, it becomes much easier to understand what is going on inside the system and why a different set of processes is a candidate for the allocation of the CPU at different time. A third unique feature of the simulator is that it allows the user to test and increase his understanding of the concepts studied by making his own scheduling decisions, through the very easy-to-use graphical user interface of the simulator. The simulator can be used by students in operating system courses or by anyone interested in learning CPU scheduling algorithms in an easier and a more effective way. 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. Duration of CPU bursts is taken from the user. The system simulates creation, concurrent execution and termination of processes. It simulates shared utilization of processor. It maintains system queues in memory. It generates reports of useful data that represent behavior of the system. The data can be stored in data base for analysis and archival purposes. It also shows comparative results of CPU scheduling algorithms empirically and graphically as well.

A user friendly and mouse driven Graphical User Interface(GUI) has been integrated .It provides the user an opportunity to save the input of all process in data base and the select the algorithm from given choice for execution. The GUI displays a Gant chart view of all processes at run time. It also provides the user an opportunity to generate a report which shows graphical as well as analytical results. It uses two tier architecture of system design. This feature makes the system attractive for experimental analysis as academic use. The system has been put through extensive experimentation. Results show 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 given workload.However, preferred treatment of short processes in SJF scheduling tends to results in increased waiting times for long processes in comparison with FCFS scheduling. Thus there is a possibility that long processes may get stranded in the ready queue because of continues arrival of shorter processes in the queue. Round Robin scheduling achieves suitable sharing of CPU and works well in time sharing environment. Shorter processes execute within a single time quantum and thus exhibit good response time. It tends to subject long processes to relatively longer response and waiting time. As the objective of this thesis work is to find the optimized solution for time sharing systems. Hence Round Robin has been optimized so that it should minimize average waiting and response time as compared to Existing Round Robin. This Optimized Round Robin also decreases the waiting and response time of longer processes. So this algorithm gives the optimized solution for time sharing systems as compared to Existing Round Robin.

SJF shows least AWT in all cases. So, it is the most efficient technique among all others. But it is not possible to find the next CPU burst. So SJF is difficult to implement. It is only the Round Robin algorithm which gives good performance for time sharing systems. It prevents all processes from starvation. But it has more Average Waiting Time (AWT) as compared to SJF. There is a need to minimize AWT of Round Robin. Enhanced Round Robin has been designed and implemented for this purpose. All the tabulated results clearly indicate that it provides lesser AWT in all cases. It’s major reason is that processes waiting in the ready queue are re-arranged according to time quantum (i.e. Processes with smaller segment will be given CPU first).This modification in the RR technique works well and gives enhanced performance for time sharing systems.

5.2 Future Work

Present system is designed to analyze the performance of existing scheduling algorithms and proposed the optimized solution for time sharing systems. Software module can be developed and analyzed for CPU scheduling in symmetric multiprocessing virtual machines. As this system allows resources of single physical machine to be shared among multiple virtual machines. Also software can be developed and integrated in the system to introduce simulation of Linux, windows 200/XP and Windows Vista scheduling policies. As multiple level queues scheduling and multiple level Feedback queue scheduling provide the scheduling policies suitable for system having variety of job running concurrently. These scheduling disciplines can be included in this system by integrating a software module that can integrate the operation of exi55sting CPU scheduling policies. Also further research work can be done on existing algorithm to find better solution for time sharing systems.

BIBLIOGRAPHY

[1] Milenkoviie M. Operating System Concepts and Design.Me Graw Hill

[2] Silberchatz A, Petorson J. L,Galvin P. B. Operating System Concepts Addison Wesley

[3] Beck L.L, System Software. Addison Wesley

[4] Tanenbaum, A modern Operating system

[5] CPU Scheduling Available at http://cs.wwwc.edu/~aabyan/352/Scheduling.html

[6] Operating System Lecture Notes CPU Scheduling. Available at

http://williamstallings.com/Extras/OS-Notes/h6.html

[7] Martin Needle's Survey of Real Time Scheduling Tools(TIK-Report No.72,November 1998)

[8] Pinedo, 1995

[9] Operating system notes available at http://choices.cs.uiuc.edu

[10] Operating System Notes Available on   [http://everything2.com/?node_id=1163115]

[11] http://www.cs.ualberta.ca/~tony/C379/Notes/]

[12] http://www.cim.mcgill.ca/~franco/OpSys-304-427/lecture-notes