TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

1. INTRODUCTION

1.1 Scheduling

"Scheduling concerns the allocation of the resources to tasks over given time period and its goal is to optimize one or more objectives"[8]. In computer science, scheduling defines a set of policies and mechanisms used in operating systems that provides the sequence in which the tasks to be done by the computer are successfully completed. It is the foundation of multiprogramming operating system. In multiprogramming systems, number of processes may run at all the time that are used to maximize the CPU utilization. In a uni-processor system, there is only a single process that can run at a time and all the remaining processes have to wait for CPU to be free and rescheduled. Scheduling is the basic function of operating system, mostly all the computer resources are scheduled for use. As CPU is one of the main and important resources of computer therefore its scheduling is essential for operating system. The major objective of scheduling is to enhance performance of the system in accordance to the criteria that are most important by the designer. A scheduler is an OS module that selects the next job to be admitted into the system and the next process to run [1].

1.2 Scheduling Levels

There are three important levels of scheduling; they are:

- **High Level Scheduling** It is also called job scheduling; it finds out which job shall be permitted to compete for the resources of the system. It is also called admission scheduling because it finds out which jobs are permitted to get admission to the system. Once jobs got admission to the system, they become processes.
- **Intermediate Level Scheduling** It finds out which processes shall be permitted to compete for the CPU. It behaves as a buffer between the admission of the jobs to the system and assigning the CPU to these jobs.
- **Low Level Scheduling** It is also called CPU scheduling. It finds out the processes that are ready for execution and assign the control of the CPU to one of them.

This level of scheduling is carried out by the dispatcher; a module that allocates CPU to the process selected using low level scheduling.

1.3 Scheduling Algorithms

When there is more than one process in processes ready queue, the operating system is supposed to decide which processes is to be run before which process. The operating system's part that makes this decision is called scheduler and this mechanism is called CPU scheduling. There are different properties of the algorithms that are used for the selection of algorithm for a given set of processes in a particular situation.

Fundamental assumptions behind most scheduling algorithms are [5, 23]:

- A set of processes in the ready queue competing for the CPU.
- Processes are independent.
- Processes are competing for set of resources.
- The job of the scheduler is to allot the CPU fairly to different processes and in a way that optimizes performance criteria.

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

- First Come First Served (FCFS)
- Shortest Job First (SJF)
- Round Robin (RR)
- Priority Scheduling
- Shortest Remaining Time First Scheduling (SRTF)

1.4 Scheduling Disciplines

Generally, there are two types of scheduling discipline; preemptive and non-preemptive. In preemptive scheduling, CPU is taken away from the running process when the process having higher priority becomes ready for execution. At the completion of higher priority process, CPU is given back to the first process. Preemptive scheduling is useful in systems in which rapid attention is given to the higher priority processes. In non-preemptive scheduling, CPU cannot be taken away from the running process until its completion. After completing execution, the process relinquishes the CPU and it is

allotted to the next process. Non-preemptive scheduling is useful in batch systems in which a selected job runs to completion.

1.5 History and Importance

Scheduling is basic function of operating system. About all computer resources have to schedule for use. Scheduling is the basis of multiprogramming. In early systems, there was only the concept of batch processing or uni-processor system in which a single process may run at a time; and all the other processes have to wait for CPU to be free. In 1950, there was no issue of CPU scheduling because of single user or batch processing. Late 1960's and early 1980's concepts of multiprogramming and time sharing was evolved. It was required to use concept of process scheduling to handle multiple users and to swap jobs to avoid idleness.

In mid 1980's and early 1990's: personal computers brought up in field, early MS- DOS and Microsoft Windows systems were non-multitasking, therefore no need of sophisticated CPU scheduling algorithms. Non-preemptive scheduler is used by Windows 3.1x. It depends on the program till end or notifies the OS that there is no need of the CPU so that it could shift to the other process. It was generally known as cooperative multitasking. Rudimentary Preemptive scheduler was introduced by Windows 95. Threads in Windows 2000 are scheduled by using Priority based and Preemptive scheduling algorithms. There are two classes of priorities: the variable class has threads that reserve priorities range from 1 to 15, and the real time class has threads that reserve priorities range from 16 to 31. Windows XP, NT and Vista uses multilevel feedback queue [2, 7].

Mac OS 9 uses cooperative scheduling. The kernel uses a Round-robin scheduling algorithm to schedule the processes. There is a copy of the thread manager for each process that is used to schedule each and every thread. Then preemptive scheduling algorithm is used by the kernel to schedule all tasks to have processor time [7].

Linux uses two separate process scheduling algorithms. One is designed for fair preemptive scheduling that is a time sharing algorithm and other is designed for real tasks. Linux scheduler is preemptive, priority based. There two classes of priorities: One is real time class ranging from 0-99 and other is for nice task level ranging from 100- 140 [7].

In FreeBSD, multilevel feedback queue having priorities range from 0-255 is used. There are four classes of priorities: interrupts reserves 0-63, top half of the kernel reserves 64-127, real-time user threads use 128-159, time-shared user threads use 160-223 and idle user threads use 224-255 [7].

In NetBSD, multilevel feedback queue having priorities range from 0-223 is used. Priorities are divided into five classes. First class is reserved for time sharing threads ranging from 0 to 63, second class for user threads ranging from 64 to 95, third class for kernel threads ranging from 96 to128, fourth class for user real-time threads ranging from 128 to 191 and fifth class for software interrupts ranging from 192 to 223 [7].

In Solaris, multilevel feedback queue having priorities range from 0-169 is used. There are four classes of priorities: the time-shared threads class has priorities from 0 to 59, the system threads class has priorities from 60 to 99, the real-time threads class has priorities from 100 to159, and the low priority interrupts class has priorities from 160 to169 [7].

1.6 Scheduling Objectives

Different objectives must be taken under consideration in the design of a scheduling discipline. Various disciplines are [6]:

- **Be fair.** A scheduling discipline use fair policy for all processes and no process can suffer indefinite postponement [2].
- **Maximize throughput.** A scheduling discipline should complete the largest number of processes per unit time.
- **Response time.** Should be minimum
- **Be predictable.** Almost same amount of time and same cost should be required to run the given job [2].
- **Minimize overhead.** Overhead is commonly considered as wasted resources. Overall system performance can be greatly be improved by using certain portion of system resources.
- **Balance resource use.** Scheduling mechanism should keep the system resources busy.

- **Enforce priorities.** Scheduling mechanism should favor the higher priority processes among processes of given priorities.
- **Better services.** Scheduling mechanism should give better services to processes that show desirable behavior.
- **Achieve a balance between response and utilization.** The best way to obtain good response time is to have efficient resources available whenever they are needed.
- **Degrade gracefully under heavy loads.** A scheduling mechanism should not collapse under the weight of a heavy system load. In one situation, when the load is heavy, scheduling mechanism should not allow new processes to be created. In other situation it should service the heavy load by providing some level of service to all processes.
- **Avoid indefinite postponement.** In many cases, indefinite postponement can be similar to deadlock. Indefinite postponement can be avoided by aging.
- **Give preference to processes holding key resources.** Sometimes a low priority process may be holding a key resource; the resource can be in demand by high priority processes. If the resource is preemptive then it is taken away from the low priority process and allocated to the high priority process. 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.
- **Policy enforcement.** Scheduling mechanism keeps track of policies that all the stated policies are carried out in case of all systems.
- **Proportionality.** Scheduling mechanism should meet user's expectation in case of interactive systems.
- **Meeting deadlines.** Scheduling mechanism should avoid losing data in case of real time system.

1.7 CPU Scheduling Algorithms Comparison Criteria

There are different criteria that have been used to compare CPU scheduling algorithms. These characteristics make significant difference in determining the best algorithm. The criteria contain the following [2, 3, 25]:

- **CPU utilization** To get the maximum CPU utilization, it is necessary to keep the CPU as busy as possible. CPU utilization varies from 0% to 100 %. In real system, it varies from 40% to 90%. For simulation purpose, CPU utilization is considered as 100%.
- **Throughput** There is a direct relation between the work completed by the CPU and CPU utilization. Number of processes that complete their execution per unit time are called throughput; it is the measure of work completed by the CPU. Algorithm provides maximum throughput is considered best algorithm.
- **Turnaround Time** The important criterion from a particular process is "how much time is required to execute that process". Turnaround time is defined as the total time required to complete execution of the process. It is the sum of the time spent waiting to get into memory, waiting into the ready queue, executing on the CPU, and doing I/O.
- **Waiting Time** It is the amount of time process has been waiting in the ready queue. Waiting time of the process can be affected by any algorithm but service time cannot. Waiting time must be kept minimum.
- **Response Time** It is the amount of time from the submission of the process to the ready queue until the process receives the first response is termed as response time. Response Time must always be kept minimum.

In addition to the above criteria, a scheduling algorithm must also have the following characteristics:

- Fairness
- Predictability
- Scalability

1.8 Scope of the Thesis

In this humble effort, it has been tried to cover at most all aspects of scheduling. Initially, different concepts and methodologies of existing scheduling algorithms have been discussed after which a new algorithm has been proposed. Having come up with the new algorithm, next endeavor is to implement the proposed algorithm and compare its

performance with the existing scheduling algorithms on the basis of simulation and results.

1.9 Thesis Objective

The objective was to study existing CPU scheduling algorithms then to propose a novel CPU scheduling algorithm. The new algorithm was then implemented and validated by comparing with other existing CPU scheduling algorithms. The comparison was based on results obtained through simulation.

1.10 Research Contribution

Proposed schemes improve existing CPU scheduling algorithms and provide efficient results on the basis of the performance criteria. Thesis work has been supported by the following Journal Papers those have been accepted and published in an International Journal:

i. Saeeda Bibi, Farooque Azam, Sameera Amjad, Wasi Haider Butt, Hina Gull, Rashid Ahmed, Yasir Chaudhry "An Efficient SJRR CPU Scheduling Algorithm" International Journal of Computer Science and Information Security (IJCSIS), Vol. 8, No. 2, 2010, pp. 222-230, ISSN: 1947-5500, Pittsburgh, PA 15213, USA

ii. Saeeda Bibi, Farooque Azam, Yasir Chaudhry "Combinatory CPU Scheduling Algorithm" International Journal of Computer Science and Information Security (IJCSIS), Vol. 8, No. 7, October 2010, pp. 39-43, ISSN: 1947-5500, Pittsburgh, PA 15213, USA

1.11 Thesis Structure

Chapter 2 discusses the Basic Concepts related to CPU scheduling. These include process, scheduling queues, schedulers, disciplines etc. Existing CPU scheduling Algorithms and their working are also explained in this chapter.

Chapter 3 provides detailed description of Proposed CPU Scheduling Algorithms such as steps, pseudo code and flow chart with examples and their performance parameters.

Chapter 4 is specifically designed for Implementation and Results of proposed algorithms and it also discusses comparison of proposed algorithms with existing algorithms on the basis of performance parameters.

Chapter 5 describes Conclusions and Future Work.

Chapter 2

2. LITERATURE SURVEY

2.1 Process

A program in execution is called a process that contains current value of the program counter, variables and registers. It is also called unit of work in computer system. The difference between program and process is that process is an activity and program is a group of instructions [9]. There are two types of processes in computer system; one is kernel process and other is user process. Kernel process is created by the kernel that looks after various system tasks and user processes are created by the user. CPU is allotted to each process by the operating system in order to perform computing work. In a uni-program operating system there is only a single user process at a time that takes the control of CPU. In a multiprogramming system, there are many independent processes at a time competing for the control of CPU [1].

Operating systems perform different activities to manage processes. These activities are [2]:

- To create and remove processes.
- To provide a means of communication among processes.
- To allocate hardware resources among processes.
- To act on exceptional conditions arise at the time of execution of the process; these are interrupts and arithmetic errors.
- To control movement of the processes i.e. to make sure that each logically enabled process makes progress towards its completion at a positive rate.

When a user job starts it execution a process is created and when the job terminates that process is destroyed. Process can be static in nature when it gives rise to one or more processes; on the other hand it is a dynamic concept that refers to a sequence of code in execution, undergoing frequent state and attribute changes.

2.1.1 Process States

The current activity of the process is called state of the process. A process can pass through series of distinct process states. Due to different events a process can change its states. Each process can be in one of these states; Start, Ready, Running, Waiting and Halted [5].

Start: The process has just arrived.

Ready: The process is ready to take the control of CPU.

Running: Control of the CPU has been assigned to the process.

Waiting: The process is doing I/O work or blocked.

Halted: The process has completed its execution and is about to leave the control of CPU.

These state names can be different across operating systems. Figure 1 shows diagram of process states.

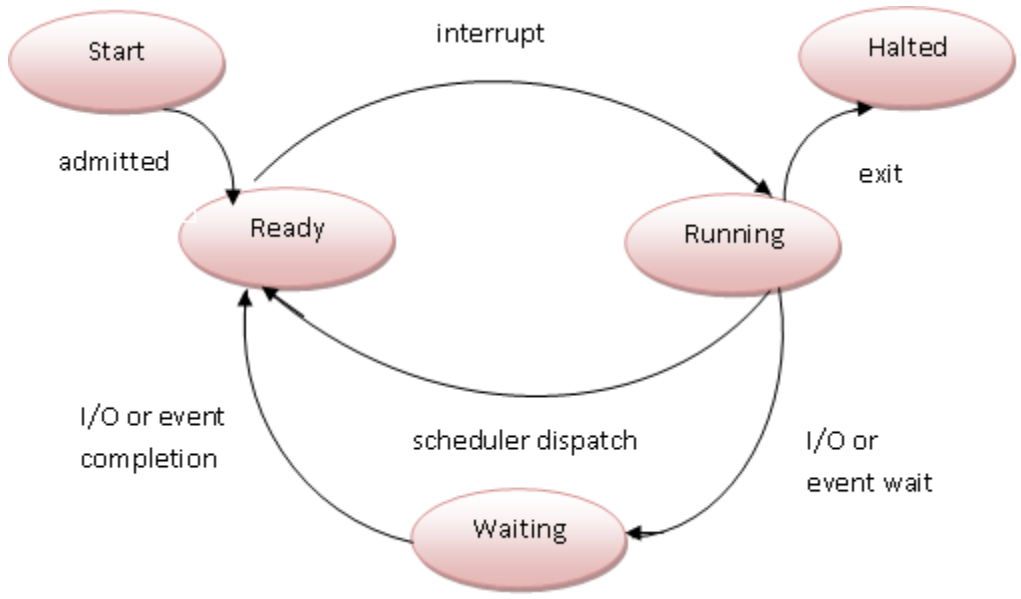


Figure 1: Diagram of Process States

2.1.2 Process Control Block

In operating system, all the information of a process is stored in Process Control Block (PCB) or a Process Descriptor. When a new process is created, the operating system creates its related PCB that is used to provide its run time description during the life time of the process. When the process terminates, its PCB is released. The PCB is a

data structure that stores various aspects of process execution and resource usage information. The information stored in PCB typically includes some or all of the following parameters [1] and these are also shown in figure 2.

- Process State
- Process ID
- Program Counter Value
- Register Values
- Memory Management Information (base/ bound registers, page tables etc)
- Processor Scheduling Information (priority, last processor burst time etc)
- I/O status information
- List of open files
- Accounting information

Process Control Block (PCB) works as a repository for any information that has different values for different processes [2].

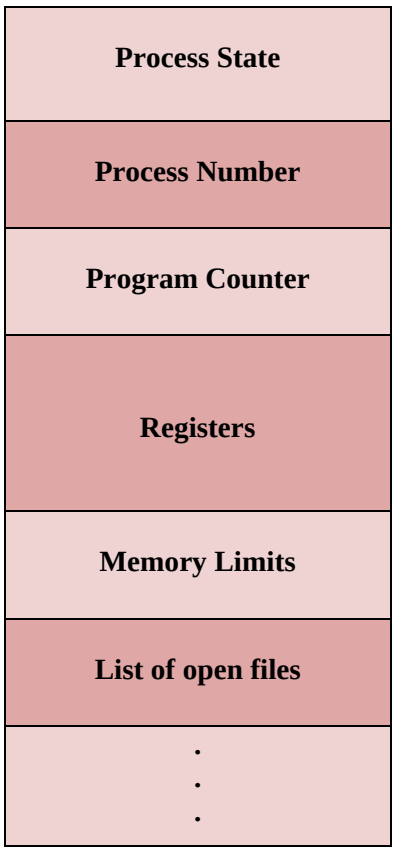


Figure 2: Process Control Block (PCB)

2.2 Scheduling Queues

Operating system maintains queues to keep record of current state of the processes. These queues are used to figure out processes in the same state that are examined by operating system resource allocation routines [5]. There are different types of queues that are used by the operating system at various states of the processes. Three different types of queues, generally needed for process scheduling [2] are listed below.

2.2.1 Job Queue

It consists of all the processes that enter into the system. All the processes that reside on secondary storage and waiting for main memory allocation are stored in job queue.

2.2.2 Ready Queue

This queue consists of those processes that reside in main memory and are ready and waiting for allocation of CPU for their execution. Generally, ready queue is stored in the form of linked list.

2.2.3 I/O Device Queues

This queue consists of all those processes that are waiting for particular I/O devices. There is a unique device queue for each device.

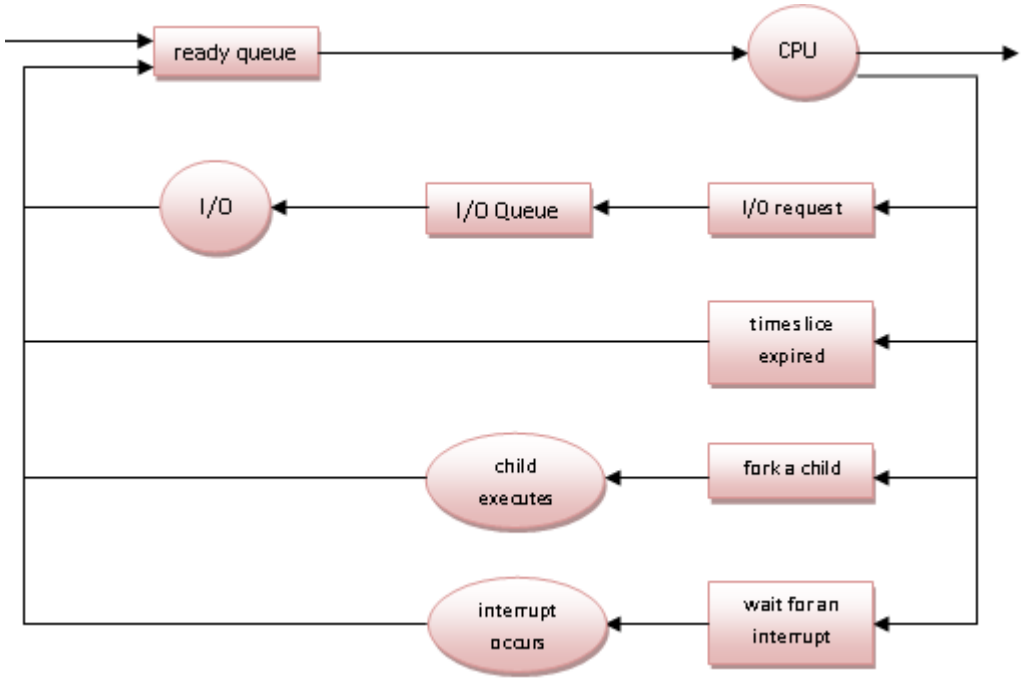


Figure 3: Queuing Diagram representation of Process Scheduling

2.3 Schedulers

Different scheduling queues come across the life time of the processes. Processes are selected by the operating system from these scheduling queues in some way. The job to be performed by the scheduler is to select processes from various queues. There are three categories of schedulers that are used in the operating system. These are

2.3.1 Long-term scheduler

The job of long-term scheduler is to select those processes from the pool of the processes that should be brought into the main memory for execution. It is also called job scheduler [2]. Long-term scheduler

- is invoked very infrequently
- controls the degree of multiprogramming
- can take more time to select a process for execution
- may be absent or use less on some systems like time sharing systems
- should do careful selection of I/O bound processes and CPU bound processes

2.3.2 Medium-term scheduler

It performs an intermediate level of scheduling. Medium-term scheduler [2]

- presents in time sharing systems
- takes away processes from main memory therefore decreases degree of multiprogramming
- improves the process-mix and change in memory requirements because of swapping-in and swapping-out of processes.

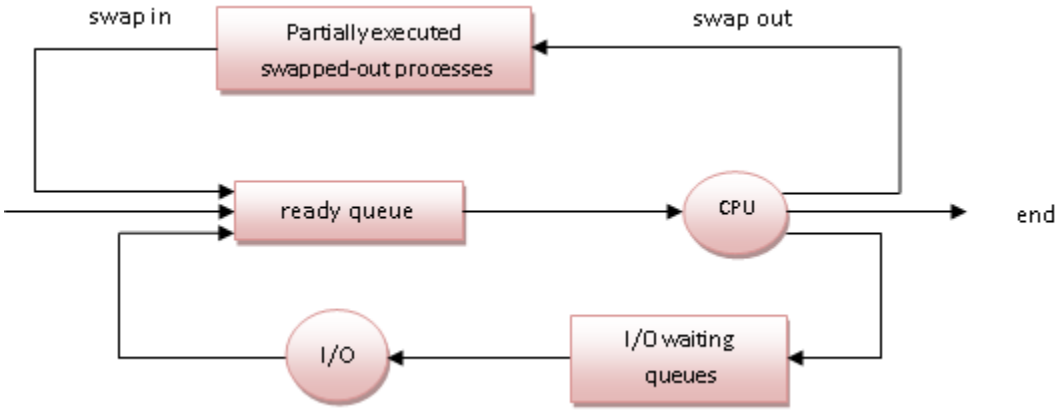


Figure 4: Diagram of Midterm Scheduler

2.3.3 Short-term scheduler

It decides to which process in main memory, CPU is allotted next for execution. It is also called CPU scheduler. Whenever any event occurs that cause to block or preempt current process; this scheduler is invoked [12]. Following are some events examples:

- clock interrupts
- I/O interrupts
- operating system calls
- signals

The main purpose of this scheduler is to maximize performance of the system in accordance with the chosen set of criteria [5]. CPU scheduler [2]

- is invoked very frequently.
- must be very fast.
- wastes some time of CPU in process scheduling.

2.4 Scheduling Disciplines

There are two categories of scheduling disciplines [25]: These are;

2.4.1 Non preemptive Scheduling

In non preemptive CPU scheduling algorithm, once the control of the CPU has been given to any process, the process keeps the control of the CPU until it has completed its execution or moved to its waiting state. In other words, when a higher priority process is waiting to get charge of the CPU, the running process is not compelled to renounce ownership of the processor. However, CPU is assigned to the other ready process when the running process becomes suspended as a result of its own action [1, 2, 12]. As this approach is intuitive therefore it is ordinary to use in process management. It also applies in those systems that invoke the scheduler without using interrupts [13]. This approach is good for “background” batch processes [18]. Examples of non preemptive scheduling algorithms are FCFS, SJF and Priority without preemption [18].

2.4.2 Preemptive Scheduling

In preemptive scheduling algorithms, CPU is allocated to the highest priority process among all processes in the ready queue. Whenever any high priority process becomes ready to execute, the current process that is using the CPU is interrupted and CPU is

allotted to that process. Similarly, when the time slice of the process expires, it may handover control of CPU to another process. Preemptive scheduling allows for better service since any one process cannot take over the control of the processor for very long [11]. Preemptive strategies provide quick response to higher priority processes or make certain fair scheduling of the CPU among all processes [13]. There is overhead of context switching in preemptive scheduling because each process rescheduling requires complete process switch [1, 2, 12]. This approach is suitable for "foreground" interactive processes [18]. Round Robin, Shortest Remaining Time First (SJF with preemption) and Priority with preemption are various examples of preemptive scheduling algorithms [18, 23].

2.5 Burst Time

The two cycles in which an executing process swaps are CPU execution and I/O waits. The time taken by the process to complete each CPU execution is called CPU burst, and the time taken by the process to wait for an I/O to fulfill its I/O request is called I/O burst. When any CPU algorithm is selected the data for burst time of processes are taken. The reason behind it is that it prevents to take the control of the CPU for those processes that have long burst time while short burst time processes are ready to run [10].

2.6 Dispatcher

It is a component of the CPU scheduling function. It is a module that is used to allocate the CPU to the process selected by the CPU scheduler. This involves [10]:

- saving state of currently running process
- loading state of selected process into registers
- executing the process at the location described by the program counter

The time taken for the dispatcher to stop one process and start another process is called dispatch latency [2].

2.7 Goals of Scheduling Algorithm for Different Systems [16, 19]

All systems

- Fairness

- Policy enforcement
- Balance

Batch systems

- Maximize CPU utilization
- Minimize turnaround time
- Maximize throughput

Interactive systems

- Proportionality
- Minimize response time

Real-time systems

- Predictability
- Meeting deadlines

2.8 CPU Scheduling Algorithms

CPU scheduling technique is used to allocate the CPU to the processes strategically on the basis of specific criteria. Different approaches are used to select processes to which control of CPU will be given. Each approach follows a scheduling algorithm for this purpose and each algorithm has its own merits and demerits. The objective of CPU scheduling is to maximize CPU utilization. For this purpose a process should be running at all the times [11]. To choose algorithm in a particular situation, properties of algorithms are used. Different performance parameters are used to compare the performance of the algorithms [1, 2, 12]. Different scheduling algorithms are described here:

2.8.1 First Come First Served (FCFS) Scheduling

First Come, First Served scheduling algorithm is the most simple and fundamental scheduling technique. Control of the CPU is assigned to the process that requests it first. The other processes are kept in the ready queue and are allocated the CPU to these processes in the way in which they arrive. FCFS is non- preemptive scheduling algorithm; process keeps the control of the CPU until it completes its execution or waits for an I/O event [10]. This algorithm is easily implemented by using FIFO queue. When the process comes into the ready queue, the PCB of the process is connected with the tail

of the ready queue. When the running process completes it execution, CPU is taken off from that process and is allocated to the process at the head of the ready queue [20]. FCFS is not appropriate for time sharing systems [14]. In FCFS, if the process having long burst time gets the control of the CPU, all the other processes having small burst time will wait in the ready queue for the CPU until that process completes its execution. This is called convoy effect [15, 16, 17]. FCFS is not used so far because its working is not good enough under any specific set of system requirements [13]. As it is simple one but there is smaller computational overhead in the execution of FCFS. FCFS scheduling algorithm gives poor performance, lower throughput, longer average waiting time and longer turnaround time. Figure 5 shows the flow chart of the FCFS algorithm:

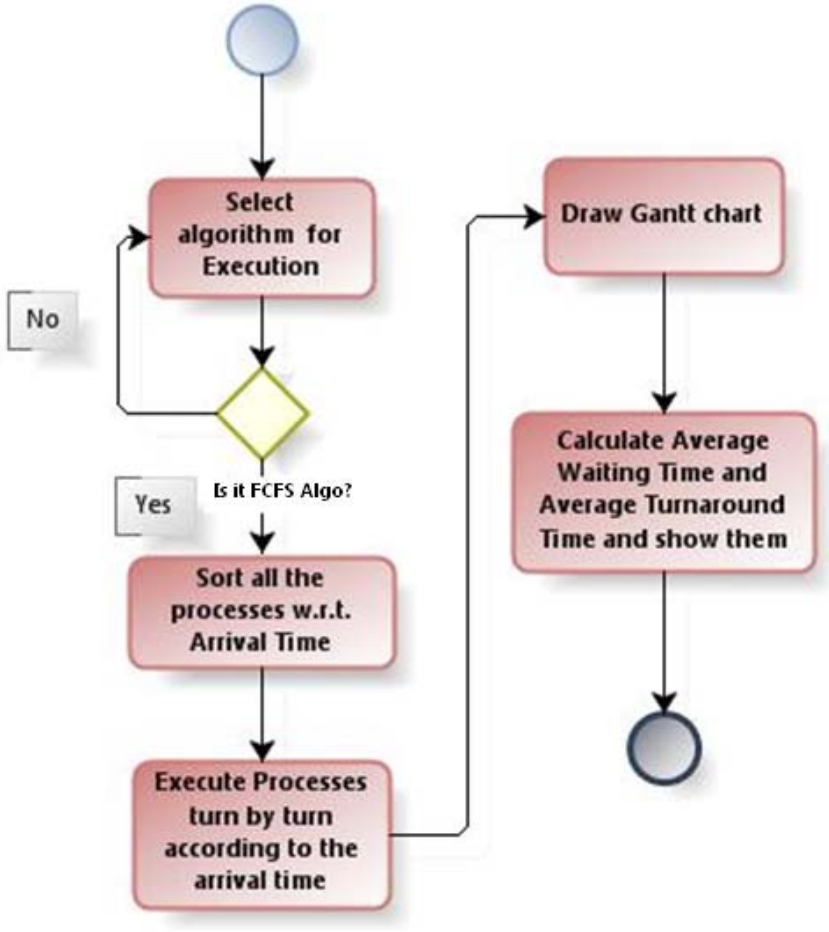


Figure 5: Flow Chart of First Come First Served Scheduling Algorithm using Simulator

Consider the following example that describes the working of FCFS algorithms. Assume the following set of processes with their burst time and arrival time:

Process Name	P1	P2	P3	P4	P5
Burst Time	10	29	3	7	12
Arrival Time	0	1	2	3	4

Arrival Time of Processes on the basis of which Sorting is performed;

0	1	2	3	4

- Total Number of processes, count= 5
- I = 4 then total passes in which processes will sort in ascending order are 4.
- As j =1; control will be at first location in given list. Here the arrival time is 0; which is compared with arrival time at second location. As 0<1, so in the next list 0 and 1 will be placed at the prior locations.

Original List

0	1	2	3	4

New List

0	1	2	3	4

- As j=2; control will be at second location in given list. The arrival time here is 1; which is compared with arrival time at third location. As 1<2, therefore their places will not be interchanged.

0	1	2	3	4

New List

0	1	2	3	4

- As j=3; control will be at third location in given list. Here the arrival time is 2; which is compared with arrival time at fourth location. As 2<3, therefore their places will not be interchanged.

0	1	2	3	4

New List

0	1	2	3	4

- As j=4; control will be at fourth location in given list. Here the arrival time is 3; which is compared with arrival time at fifth location. As 3<4, so in the next list 3 and 4 will be placed at the prior locations.

0	1	2	3	4

New List

0	1	2	3	4

- Now I = 3, again iteration will start for j loop, it will check first four values whether they are in ascending order or not. If they are not in ascending order then arrange them in that order.

- Now I = 2, iteration will start third time for j loop and check values of first three locations whether they are in ascending order or not. If they are in that order, they remain left.

- Now I = 1, iteration will start fourth time for j loop and check the values of first two locations whether they are in ascending order or not. If they are in ascending order they remain at the places where they are.

- The **resultant or final list** will be

0	**1**	**2**	**3**	**4**

- According to the arrival time's arrangement, process name and their burst times are also arranged.

Process Name	**P1**	**P2**	**P3**	**P4**	**P5**
Burst Time	10	29	3	7	12
Arrival Time	0	1	2	3	4

Sorting is completed here. Next step is to find the waiting time of each process.

- Find waiting time of each process;

  Waiting time of P1, wtime[0] = 0
  Waiting time of P2, wtime[1] = 10
  Waiting time of P3, wtime[2] = 39
  Waiting time of P4, wtime[3] = 42
  Waiting time of P5, wtime[4] = 49

- Last step is to calculate total waiting time, average waiting time and average turnaround time;

  Total Waiting Time (tw_time) = 140
  Average Waiting Time (avg_wt) = 28
  Average Turnaround Time (avg_tatime) = 40.2

Gantt chart of above example

0 10	39	42	49	61
P1	**P2**	**P3**	**P4**	**P5**

Graphical Representation of FCFS Gantt chart

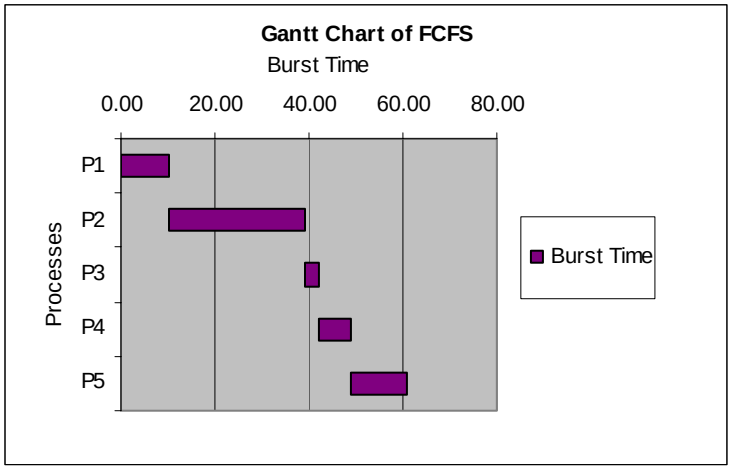


Figure 6: Graphical Representation of FCFS Gantt chart

2.8.2 Shortest Job First (SJF) Scheduling

Shortest Job First scheduling technique is a technique in which processes having smallest burst time are executed first [13]. If processes have same burst time then FCFS technique is used for execution of processes. SJF may be either preemptive or non-preemptive. The choices are made when new process is arrived at the ready queue while prior process is executing. If new process has smallest CPU burst time as compared to the currently executing process then a preemptive SJF technique will preempt the currently

executing process while non-preemptive SJF will not preempt the currently executing process and allow the process to finish it CPU burst. Preemptive SJF is also known as Shortest Remaining Time First (SRTF) [1, 2].

SJF scheduling is optimal because it provides minimum average waiting time, average turnaround time [4, 15] but at the level of short term CPU scheduler it cannot be implemented [2]. The main problem of SJF is to find out which of the presently running process has shortest burst time [4, 24]. One approach is to compute approximate length of the next CPU burst, in this way we can find the process with shortest predicted burst time [2]. Generally, exponential average of the measured lengths of previous CPU bursts can be used to predict next CPU burst. Here is the formula that defines an exponential average for α , $0 \leq \alpha \leq 1$;

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

Here, t_n be the length of the nth CPU burst and τ_{n+1} be our predicted value for the next CPU burst.

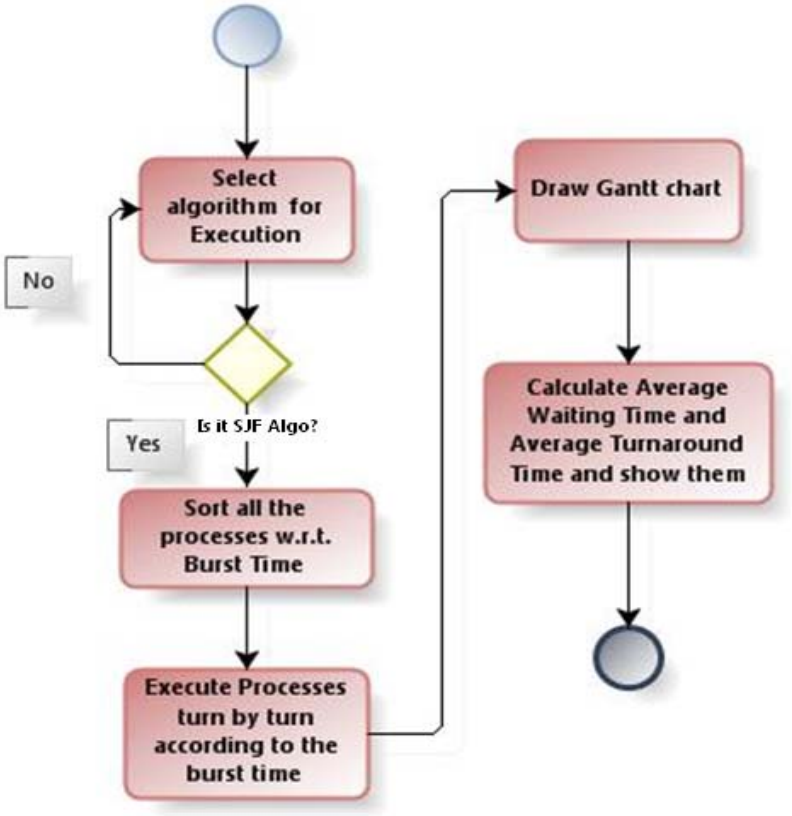


Figure 7: Flow Chart of Shortest Job First Scheduling Algorithm using Simulator

Non-preemptive SJF technique is not suitable for time sharing systems [6] while SRTF is suitable for these systems. There is also a problem of starvation for long processes in the SJF [13]. SRTF has higher overhead than SJF non-preemptive. SRTF has to keep track of the elapsed service time of the running job and has to handle occasional preemptions. Flow chart of SJF is shown above in Figure 7:

Following example describes the working of Shortest Job First Scheduling algorithm. Assume the following set of processes with their burst time and arrival time:

Process Name	**P1**	**P2**	**P3**	**P4**	**P5**
Burst Time	10	29	3	7	12
Arrival Time	0	1	2	3	4

Burst Time of Processes on the basis of which Sorting is performed;

10	29	3	7	12

- Total Number of processes, count= 5
- I = 4 then total passes in which processes will sort in ascending order are 4.
- As j =1; control will be at first location in given list. Here the burst time is 10; which is compared with burst time at second location. As 10<29, so in the next list 10 and 29 will be placed at the prior locations.

Original List

10	29	3	7	12

New List

10	29	3	7	12

- As j=2; control will be at second location in given list. The burst time here is 29; which is compared with burst time at third location. As 29>3, therefore their places will be interchanged.

10	29	3	7	12

New List

10	3	29	7	12

- As j=3; control will be at third location in given list. Here the burst time is 29; which is compared with burst time at fourth location. As 29>7, therefore their places will also be interchanged.

10	3	29	7	12

New List

10	3	7	29	12

- As j=4; control will be at fourth location in given list. Here the burst time is 29; which is compared with burst time at fifth location. As 29>12, so in the next list 29 and 12 will be interchanged.

10	3	7	29	12

New List

10	3	7	12	29

- Now I = 3, again iteration will start for j loop, it will check first four values whether they are in ascending order or not. If they are not in ascending order then arrange them in that order. Then the list will become;

3	7	10	12	29

- Now I = 2, iteration will start third time for j loop and check values of first three locations whether they are in ascending order or not. If they are in that order, they remain left.

- Now I = 1, iteration will start fourth time for j loop and check the values of first two locations whether they are in ascending order or not. If they are in ascending order they remain at the places where they are.

- The **resultant or final list** will be

3	**7**	**10**	**12**	**29**

- According to the burst time's arrangement, process name and their arrival time are also arranged.

Process Name	P3	P4	P1	P5	P2
Burst Time	3	7	10	12	29
Arrival Time	2	3	0	4	1

Sorting is completed here. Next step is to find the waiting time of each process.

- Find waiting time of each process;

  Waiting time of P3, wtime[0] = 0
  Waiting time of P4, wtime[1] = 3
  Waiting time of P1, wtime[2] = 10
  Waiting time of P5, wtime[3] = 20
  Waiting time of P2, wtime[4] = 32

- Last step is to calculate total waiting time, average waiting time and average turnaround time;

  Total Waiting Time (tw_time) = 65
  Average Waiting Time (avg_wt) = 13
  Average Turnaround Time (avg_tatime) = 25.2

Gantt chart of above example

0 3	10	20	32	61
P3	**P4**	**P1**	**P5**	**P2**

Graphical Representation of SJF Gantt chart

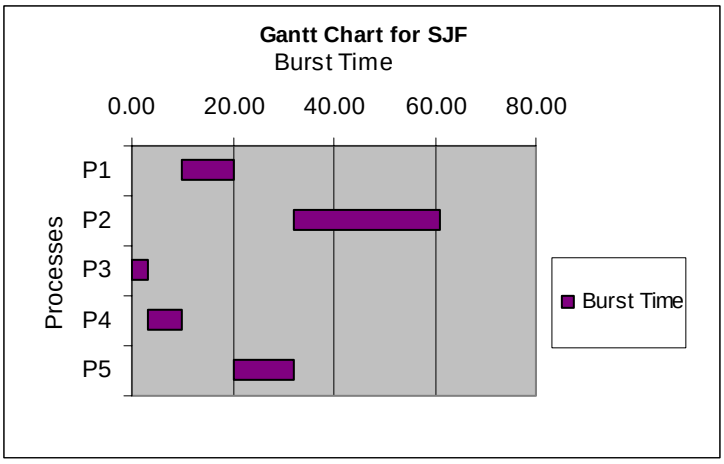


Figure 8: Graphical Representation of SJF Gantt chart

2.8.3 Round Robin (RR) Scheduling

One of the oldest, simplest and most widely used scheduling algorithms is Round Robin Scheduling algorithm. It is specially designed for time sharing system. The working of RR is similar to FCFS, but in this algorithm to switch between processes preemption is added [2]. A small unit of time called time slice or time quantum is assigned to each process in this algorithm. All the ready processes are placed in the queue. The CPU scheduler allocates CPU to the first process from the ready queue for the time interval of assigned quantum. New coming processes are added to the tail of the queue [30]. Here two situations occur; first is that when a process has completed its task before the expiry of time quantum, its release the CPU voluntarily and is assigned to the next process in the ready queue [26]. Second situation is that if the CPU burst of presently running process is more than time quantum and the time quantum expires then the control of the CPU is forcefully taken from that process [21]. Here context switching occurs and the process will be stored at the tail of the ready queue [2]. In both cases, the CPU scheduler allocates the control of CPU to the process next in the ready queue.

If there are **n** processes present in the ready queue and the time slice given to each process is **q**, then preferably each process would take 1/n of the CPU time in chunks of **q** time units, and each process would wait no longer than **(n-1)q** time units until its next time slice [2].

The Round-Robin algorithm's performance entirely depends on the size of the quantum [17, 20, 24]. If the time quantum is too little then it causes a number of context switches that affects the CPU efficiency. If the time quantum is too long then it causes poor response time and estimates FCFS [24, 27, 29]. Single time quantum is required to short processes for execution while several time quanta are used for execution of long processes [5]. Size of the time quantum also affects the turnaround time [2]. In RR algorithm average waiting time is high so deadlines are rarely met in RR policy [29]. Here is a flow chart of RR in figure 9;

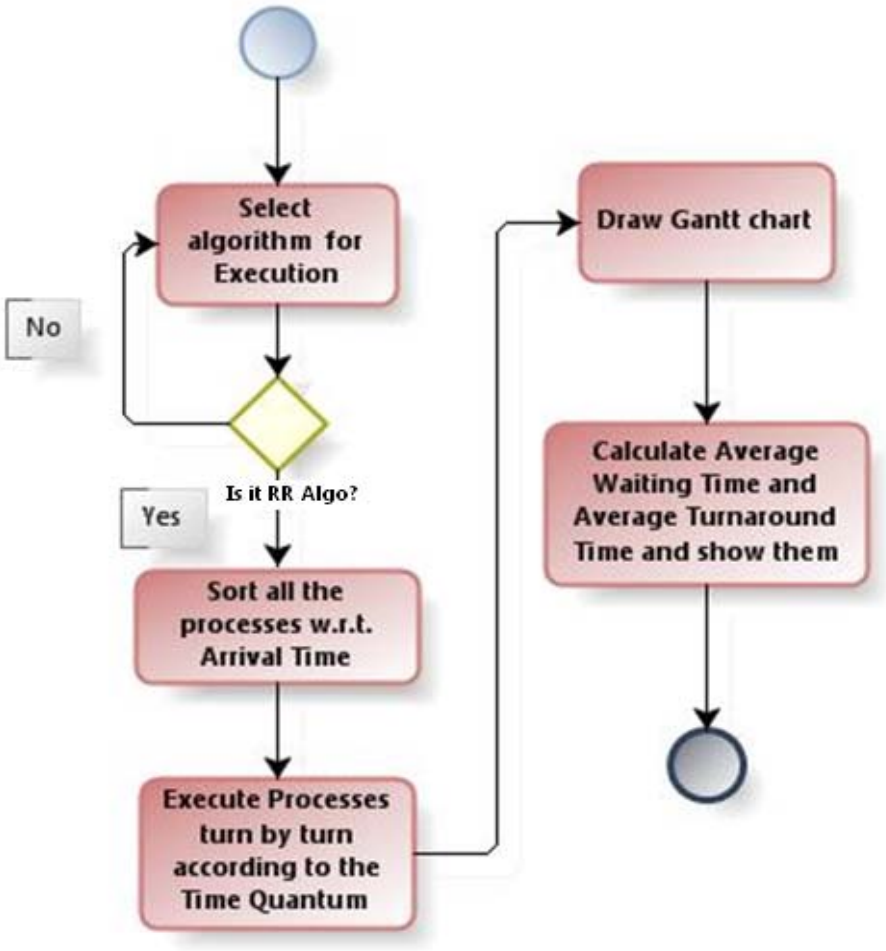


Figure 9: Flow Chart of Round Robin Scheduling Algorithm using Simulator

Given is the example of Round Robin Scheduling Algorithm that describes its working. Assume the following set of processes with their burst time and arrival time:

Process Name	P1	P2	P3	P4	P5
Burst Time	10	29	3	7	12
Arrival Time	0	1	2	3	4

Assume Time Quantum entered by the user is 10.

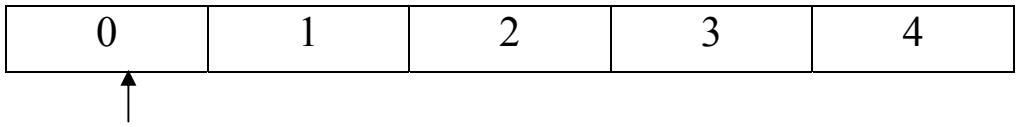
Arrival Time of Processes on the basis of which Sorting is performed;

0	1	2	3	4

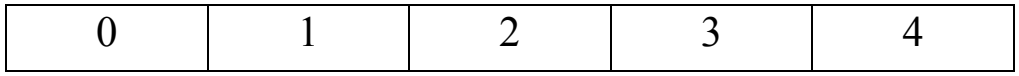
- Total Number of processes, count= 5

- I = 4 then total passes in which processes will sort in ascending order are 4.
- As j =1; control will be at first location in given list. Here the arrival time is 0; which is compared with arrival time at second location. As 0<1, so in the next list 0 and 1 will be placed at the prior locations.

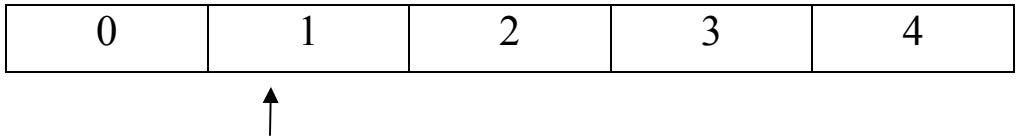
Original List



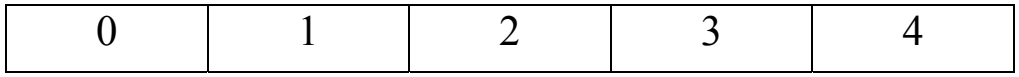
New List



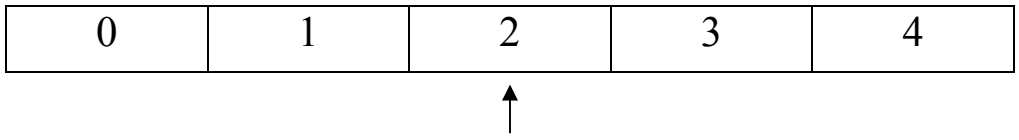
- As j=2; control will be at second location in given list. The arrival time here is 1; which is compared with arrival time at third location. As 1<2, therefore their places will not be interchanged.



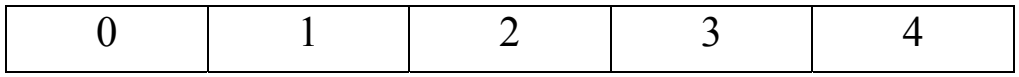
New List



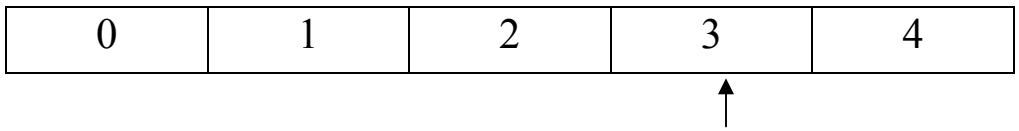
- As j=3; control will be at third location in given list. Here the arrival time is 2; which is compared with arrival time at fourth location. As 2<3, therefore their places will not be interchanged.



New List



- As j=4; control will be at fourth location in given list. Here the arrival time is 3; which is compared with arrival time at fifth location. As 3<4, so in the next list 3 and 4 will be placed at the prior locations.



New List

0	1	2	3	4

- Now I = 3, again iteration will start for j loop, it will check first four values whether they are in ascending order or not. If they are not in ascending order then arrange them in that order.

- Now I = 2, iteration will start third time for j loop and check values of first three locations whether they are in ascending order or not. If they are in that order, they remain left.

- Now I = 1, iteration will start fourth time for j loop and check the values of first two locations whether they are in ascending order or not. If they are in ascending order they remain at the places where they are.

- The **resultant or final list** will be

0	**1**	**2**	**3**	**4**

- According to the arrival time's arrangement, process name and their burst times are also arranged.

Process Name	**P1**	**P2**	**P3**	**P4**	**P5**
Burst Time	10	29	3	7	12
Arrival Time	0	1	2	3	4

Sorting is completed here.

- Next step is to find out the process having maximum burst time.
- Now calculate the maximum number of time longer burst time process will execute.

  Dim = (29/10) +1 = 3.9 =3

- Next step is to allocate '0' value to arrays used to save the executed time of the processes.

  R_robin[0,0] = 0
  r_robin[0,1] = 0
  r_robin[0,2] = 0
  r_robin[1,0] = 0
  r_robin[1,1] = 0
  r_robin[1,2] = 0

r_robin[2,0] = 0
r_robin[2,1] = 0
r_robin[2,2] = 0
r_robin[3,0] = 0
r_robin[3,1] = 0
r_robin[3,2] = 0

- Next find executed time, remaining burst time of processes and value of counter that counts how many times a process will execute.
  - For first process,
    r_robin[0,0] = 10, btime[0] = 0 , counter[0] = 0
  - For second process,
    r_robin[1,0] = 10, btime[1] = 19
    r_robin[1,1] = 10, btime[1] = 9
    r_robin[1,2] = 9, btime[1] = 0, counter[1] = 2
  - For third process,
    r_robin[2,0] = 3, btime[2] = 0 , counter[2] = 0
  - For fourth process,
    r_robin[3,0] = 7, btime[3] = 0 , counter[3] = 0
  - For fifth process,
    r_robin[4,0] = 10, btime[4] = 2
    r_robin[4,1] = 2, btime[4] = 0 , counter[4] = 1

- Second last step is to find waiting time of each process;
  Waiting Time of P1, wtime[0] = 0
  Waiting Time of P2, wtime[2] = 32
  Waiting Time of P3, wtime[3] = 20
  Waiting Time of P4, wtime[4] = 23
  Waiting Time of P5, wtime[5] = 40

- Last step is to calculate total waiting time, average waiting time and average turnaround time;
  Total Waiting Time (tw_time) = 115
  Average Waiting Time (avg_wt) = 23
  Average Turnaround Time (avg_tatime) = 35.2

Gantt chart of above example

0 10	20	23	30	40	50	52	61
P1	**P2**	**P3**	**P4**	**P5**	**P2**	**P5**	**P2**

Graphical Representation of RR Gantt chart

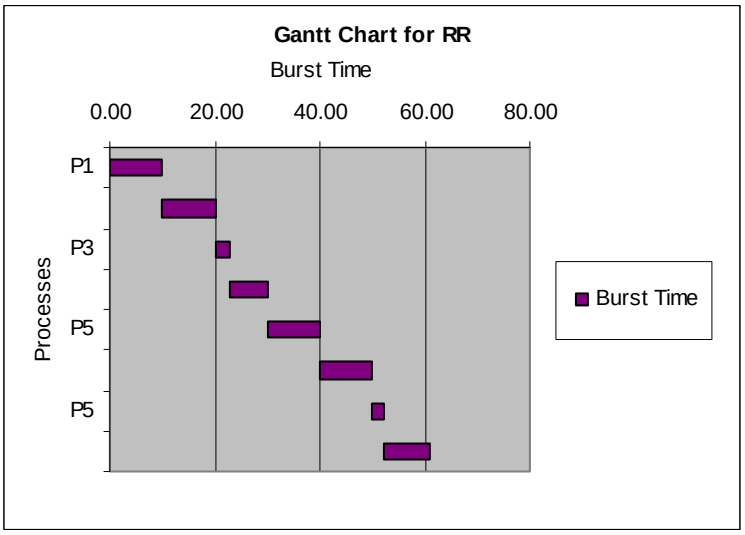


Figure 10: Graphical Representation of RR Gantt chart

2.8.4 Priority Scheduling

In priority scheduling, externally assigned priority is used to allocate control of CPU to the processes [13]. Process having higher priority takes the control of the CPU first and then other processes having low priorities will take it. Processes having equal priorities are executed on the basis of FCFS approach [21]. Priority can be assigned dynamically or statically [4]. Generally there are fixed range of numbers that are used for priorities; such as 0 to 7 or 0 to 4,095. There is no general agreement on lowest or highest priorities. Some systems take low numbers as a highest priority and some take greater number as highest priority. There are two ways to define priorities; these are internal and external. In internally defined priorities there is some measurable quantity or quantities that are used to compute the priority of a process and in externally defined priorities criteria external to the operating system are set [2]. Priority scheduling may be preemptive or non-preemptive. In non-preemptive priority scheduling, the running process will not relinquish the control of CPU until it completes its execution. All the other processes whether they have higher priorities or low priorities have to wait in the ready queue for the processor. In preemptive priority scheduling, priorities of all incoming processes are checked and if any process comes that have priority higher than the running process, the running process has to preempt the CPU and CPU is allocated to the higher priority process [2]. The major problem with this algorithm is indefinite blocking or starvation in which low priorities processes wait indefinitely for the CPU [24]. This problem can be

solved by using a technique called aging [24] in which priority of the lower processes waiting for the CPU for long time is gradually increased [20]. In figure 11, flow chart of priority scheduling algorithm is shown:

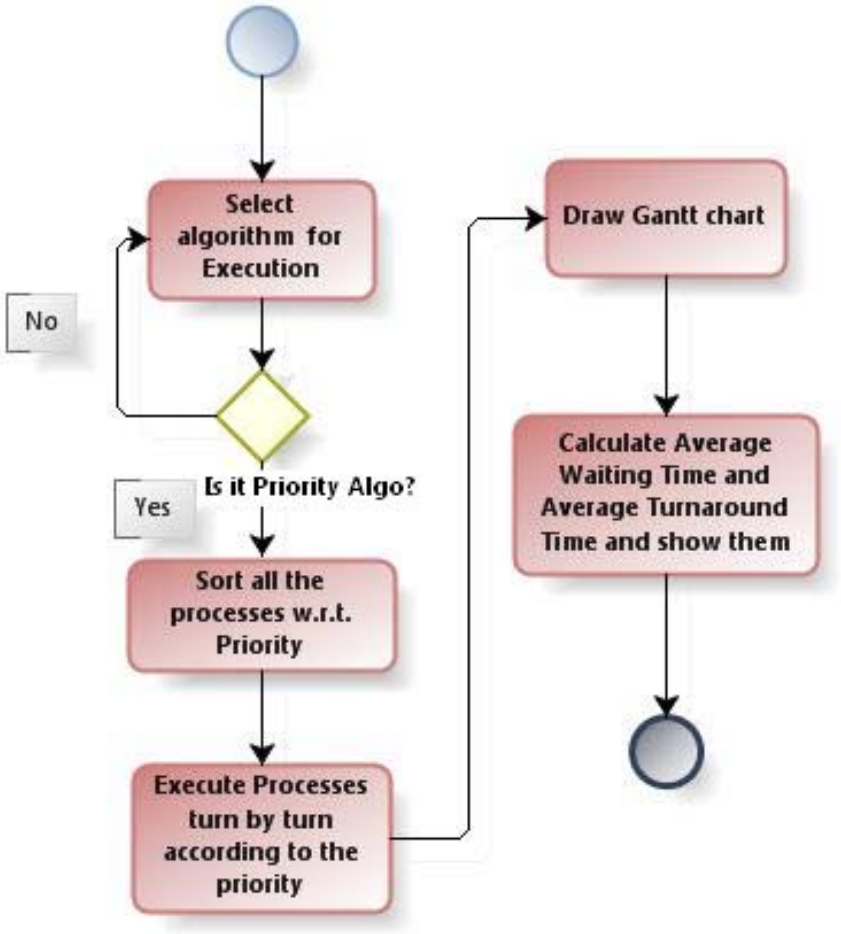


Figure 11: Flow Chart of Priority Scheduling Algorithm using Simulator

In the following example, working of priority scheduling algorithm is described. Assume the following set of processes with their burst time and arrival time:

Process Name	P1	P2	P3	P4	P5
Burst Time	10	29	3	7	12
Arrival Time	0	1	2	3	4
Priority	6	8	2	1	4

Priority of Processes on the basis of which Sorting is performed;

6	8	2	1	4

- Total Number of processes, count= 5
- I = 4 then total passes in which processes will sort in ascending order are 4.
- As j =1; control will be at first location in given list. Here the priority is 6; which is compared with priority at second location. As 6<8, so in the next list 6 and 8 will be placed at the prior locations.

Original List

6	8	2	1	4

New List

6	8	2	1	4

- As j=2; control will be at second location in given list. The priority here is 8; which is compared with priority at third location. As 8>2, therefore their places will be interchanged.

6	8	2	1	4

New List

6	2	8	1	4

- As j=3; control will be at third location in given list. Here the priority is 8; which is compared with priority at fourth location. As 8>1, therefore their places will also be interchanged.

6	2	8	1	4

New List

6	2	1	8	4

- As j=4; control will be at fourth location in given list. Here the priority is 8; which is compared with priority at fifth location. As 8>4, so in the next list 8 and 4 will also be interchanged.

6	2	1	8	4

New List

6	2	1	4	8

- Now I = 3, again iteration will start for j loop, it will check first four values whether they are in ascending order or not. If they are not in ascending order then arrange them in that order. Then the list will become;

2	1	4	6	8

- Now I = 2, iteration will start third time for j loop and check values of first three locations whether they are in ascending order or not If they are not in ascending order then arrange them in that order. Then the list will become;

1	2	4	6	8

- Now I = 1, iteration will start fourth time for j loop and check the values of first two locations whether they are in ascending order or not. If they are in ascending order they remain at the places where they are.

- The **resultant or final list** will be

1	**2**	**4**	**6**	**8**

- According to the priority's arrangement, process name and their arrival time are also arranged.

Process Name	**P4**	**P3**	**P5**	**P1**	**P2**
Burst Time	7	3	12	10	29
Arrival Time	3	2	4	0	1
Priority	1	2	4	6	8

Sorting is completed here. Next step is to find the waiting time of each process.

- Find waiting time of each process;

Waiting time of P4, wtime[0] = 0
Waiting time of P3, wtime[1] = 7
Waiting time of P5, wtime[2] = 10
Waiting time of P1, wtime[3] = 22
Waiting time of P2, wtime[4] = 32

- Last step is to calculate total waiting time, average waiting time and average turnaround time;

  Total Waiting Time (tw_time) = 71
  Average Waiting Time (avg_wt) = 14.2
  Average Turnaround Time (avg_tatime) = 26.4

Gantt chart of above example

0 7	10	22	32	61
P4	**P3**	**P5**	**P1**	**P2**

Graphical Representation of Priority Gantt chart

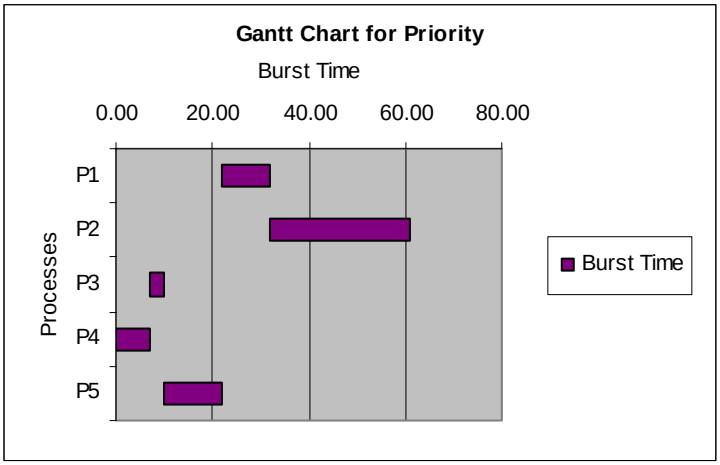


Figure 12: Graphical Representation of Priority Gantt chart

2.8.5 Multiple-Level Queues (MLQ) Scheduling

|All of the above scheduling disciplines are used for particular applications. But the system in which different jobs are running concurrently good performance cannot be obtained if the system uses only one scheduling discipline. This problem can be solved by combining several scheduling algorithms in one system. This technique is called multilevel queue scheduling. In multilevel queue scheduling ready queue is separated into different number of queues. Each queue in multilevel queues scheduling has its own scheduling algorithm [10]. These queues are separated on the basis of foreground

processes and background processes. Round Robin (RR) algorithm is used to schedule foreground queue while First Come First Served (FCFS) algorithm is used to schedule background processes. Processes are permanently assigned to the specific queue based on their properties for example process type, process priority or memory size [2, 27]. The scheduling among the queues is implemented as fixed priority preemptive scheduling. For example, foreground queue may have higher priority than background queue. Figure 13 shows multilevel queue scheduling [2].

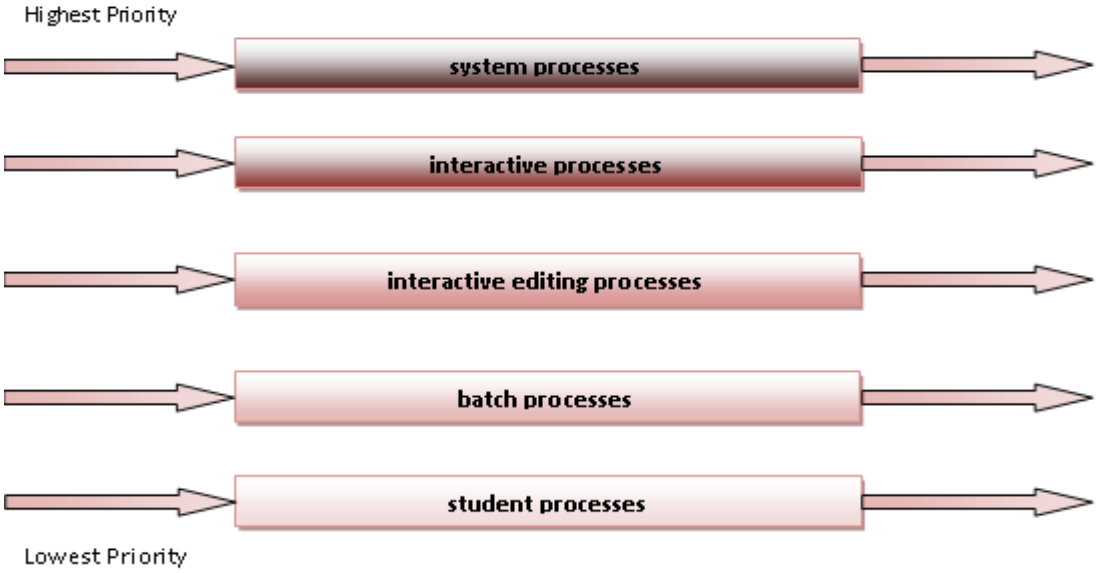


Figure 13: Multilevel Queue Scheduling

Processes might be divided into the following classes [2].

- High priority system processes
- Interactive programs
- Interactive editing processes
- Batch jobs
- Student processes

The subdivision of above classes may also exist. For example system processes can be subdivided into these [20]:

- 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. All the processes of highest priority queue are serviced first then processes of next

queue are served using its own scheduling technique. We can say that it is a preemptive priority type scheduling discipline among the queues. When all the processes of higher priority queues are completely executed, the process from the lowest priority queue may be selected for execution. This process can be preempted by arrival of a process in one of the higher priority queues [1, 12]. The other type of scheduling among the queues is to time slice between the queues in which each queue is given a certain amount of CPU time to schedule its processes [2].

2.8.6 Multilevel Feedback Queue Scheduling

It is similar to multilevel queue scheduling but it allows processes to move between queues depending upon its CPU time [27]. Higher priority queues keep the processes that are I/O-bound and interactive processes while lower priority queues hold the processes requiring larger CPU-time [2, 15]. Processes from higher priority, non empty ready queue are selected by the scheduler when CPU is available. It uses round robin scheduling within the queues. Priority of the processes is adjusted by the scheduler dynamically [22]. Problem of starvation can be removed by moving processes in the lower priority queue into the higher priority queue. This scheme is called aging [10]. There are some parameters that are used to define multilevel feedback queue. These are [28];

- Number of queues
- Scheduling algorithms for each queue
- Conditions for increasing or decreasing a process priority
- Methods used to decide which queue a process will enter first time

Figure 14 shows the working of multilevel feedback queues.

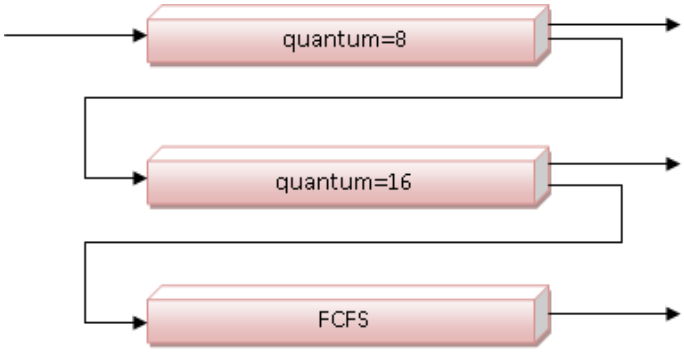


Figure 14: Multilevel Feedback Queue Scheduling

Multilevel Feedback Queue Scheduling is the most general as well as the most complex scheduling technique.

2.9 Additional Algorithms

2.9.1 Fixed Priority Scheduling Algorithm

In this algorithm, initially a priority level that is permanent and unchangeable is assigned to each process. It is easy to understand, manage and implement but it does not remove starvation. A low priority process will always have to wait for higher priority processes to relinquish the control of CPU and its priority always remains low [10].

2.9.2 Rate Monotonic Scheduling Algorithm

This scheduling algorithm is used in real time operating systems having static priority class of scheduling. The priorities are assigned on the basis of duration of execution time of the processes; shorter the duration of execution, higher will be priority of that process [30].

The main problem with this algorithm is that CPU can never be used 100 percent completely when we use this algorithm. Its main advantage is that it is optimal as compared to fixed priority scheduling algorithms [10].

2.9.3 Rate Monotonic with Delayed Preemption Scheduling Algorithm

The central problem with Rate Monotonic is that there are a large number of context switches in it which may be very expensive. The Rate Monotonic with delayed Preemption reduces the number of context switches by allowing the process in execution to delay in yielding the CPU to the higher priority processes [10].

2.9.4 Deadline Monotonic Scheduling Algorithm

In this algorithm priority of the process is calculated by the time interval between the starting time of the process and the deadline of the process. Processes having shorter interval of time are given higher priorities while processes having larger interval of time are given lower priorities. This algorithm enforces the constraints to systems that each process deadline must be less than or equal to its period [10].

2.9.5 Dynamic Priority Scheduling Algorithm

In this algorithm, changing can be made in the priority of the process; the reason that makes it more complex and considerably more powerful than other algorithms. In this algorithm starvation can be removed by using aging of the processes. These types of algorithms are very helpful in real time systems. If any process wants to complete its execution and its deadline is coming then the priority of that process can be increased so that it can be able to complete timely. When there is heavy workload in the system, it will be very difficult to predict dynamic algorithms [10].

2.9.6 Earliest Dead Line First Scheduling Algorithm

This algorithm uses the priority that is calculated on the basis of the deadline of the process. Highest priority is given to the process that has shortest remaining time when its deadline is scheduled. It is a preemptive scheduling algorithm that forcefully takes the control of CPU from the currently running low priority process if any higher priority process is ready to run. Its CPU utilization is not greater than 100 percent [10].

2.9.7 Least Slack Time First Scheduling Algorithm

Each process uses the priority based upon its slack time. The slack time of a process is calculated by using this formula

Slack time = absolute deadline – current time – remaining compute time.

Process with greater slack time has lower priority while process with smaller slack time has higher priority. A large number of context switches are produced in this algorithm and it always needs knowledge for execution time of tasks [10].

2.10 Summary

In this chapter an overview of the well known CPU scheduling techniques is provided. All the necessary details of existing algorithms, their advantages and disadvantages have been discussed. It is essential to mention here that the proposed SJRR CPU scheduling algorithm is an improvement in Round Robin scheduling algorithm. Similarly, Combinatory CPU scheduling algorithm is somehow extension of Shortest Job First scheduling algorithm. Both of these proposed algorithms and their comparisons with existing algorithms will be discussed in detail in the next chapters.

Chapter 3

3. PROPOSED CPU SCHEDULING ALGORITHMS

In this chapter proposed CPU scheduling algorithms "Shortest Job Round Robin (SJRR)" and "Combinatory Algorithm" are presented. SJRR Algorithm sorts all incoming processes according to their burst time in ascending order in the ready queue and then uses the time quantum to execute processes. In Combinatory Algorithm, a new factor F is suggested with each process. This factor adds two basic factors (arrival time and burst time) of all incoming processes and then sorts all of the processes according to this factor in ascending order. After sorting, execution of the processes is performed one by one. Details, methodology and performance parameters are discussed in this chapter.

3.1 Introduction

Proposed SJRR CPU scheduling algorithm is based on sorting of processes with respect to burst time in ascending order in the ready queue and then utilizing time quantum to allocate processes to the CPU for execution. At the core of the proposed algorithm work two well known and well practiced algorithms namely Shortest Job First scheduling algorithm and the Round Robin scheduling algorithm.

Proposed Combinatory CPU scheduling algorithm adds two basic factors of processes that are arrival time and burst time and store it into a new factor F. Processes are sorted in ascending order in the ready queue according to this factor and after that execution of processes start in the ascending order. Proposed scheduling algorithms are efficient and have better performance as compared to other CPU scheduling techniques which has been discussed in the upcoming sections.

3.2 Explanation of SJRR and Combinatory Algorithms

SJRR scheduling algorithm is based on two existing scheduling algorithms, Shortest Job First algorithm and Round Robin algorithm. Shortest Job First algorithm is non-preemptive technique that executes smaller burst time processes first and longer processes at the end. Round Robin algorithm is preemptive in nature and uses the concept of time quantum to execute processes. In proposed SJRR algorithm, an efficient sorting technique is used to sort the number of processes according to their burst time in

ascending order in the ready queue. The time quantum is taken as mid value of the burst time of the sorted list of the processes. If there are odd numbers of processes in the ready queue then the burst time of middle most process is used as a time quantum otherwise take the average of the burst time of the two middle most processes in the queue. This time quantum is used to execute the processes like Round Robin Scheduling algorithm. The calculated time quantum is given to the first smallest burst time process in the ready queue for execution. Upon the completion of the time quantum, the process is preempted and CPU control is transferred to the next smaller burst time process in the ready queue. In this way, time quantum is assigned to each process in the ready queue for its execution. Each process will execute according to the given time quantum. If any process completes its execution in the given time quantum then it gets off the CPU and control of the CPU is transferred to the next process in the queue otherwise that process will be preempted and placed at the end of the ready queue.

Combinatory scheduling algorithm is based on a new factor F that is calculated by taking sum of the two basic factors (arrival time and burst time). Here is the equation that summarizes this relation:

F= Arrival Time + Burst Time

On the basis of this new factor F; sorting is performed that arrange the processes in ascending order in ready queue i.e. the processes having lowest value of factor F are placed at start of the ready queue and the processes having highest value of the processes are placed at end of the queue. The process having lowest value of factor F will execute first and the process having highest value of factor F will execute at last. Depend on this new factor CPU executes the process that:

- Has shortest burst time
- Submit to the system at start

There are a number of factors used to measure the performance of the CPU scheduling algorithms. These include CPU utilization, waiting time, turnaround time, throughput, response time, fairness, context switching and starvation etc. CPU utilization, waiting time, turnaround time, throughput and response time are low in first come first served algorithm because longer processes can hog the processor. This algorithm does not provide fairness between processes. As shortest job first algorithm minimizes waiting time and turnaround time, maximizes throughput and response time; therefore it is

optimal. But there is a problem of starvation in SJF because longer processes have to wait for a long time if upcoming processes have shorter burst time as compared to the present processes so it does not provide fairness between processes. There is a problem of context switching in round robin scheduling algorithm especially with the small time unit but if the time unit is greater then it works like first come first served. In priority based algorithm, processes having higher priorities are executed first. In this algorithm, problem of aging occurs.

Proposed SJRR CPU scheduling algorithm not only gives minimum waiting time, minimum turnaround time and also maximizes CPU utilization. As it provides fairness between processes therefore no starvation occurs in it and it is suitable for time sharing system. There is no more overhead of context switching in this proposed scheduling algorithm like round robin scheduling algorithm.

Proposed Combinatory CPU scheduling algorithm gives minimum waiting time, minimum turnaround time and maximizes CPU utilization as compared to the existing CPU scheduling algorithms and proposed SJRR CPU scheduling algorithm. The problem of starvation has been resolved at much more extent in this algorithm. It also resolves the problem of context switching completely. In the upcoming sections, the steps, pseudo code and flow chart of the proposed scheduling algorithms are presented that are used for better understanding of new algorithms. Simple examples are also described here that depict the actual working of the proposed algorithms.

3.3 Steps of SJRR Scheduling Algorithm

1. Take list of processes, their burst time, arrival time and time quantum.
2. Arrange processes and their relative burst time in ascending order using any sorting technique.
3. Iterate through the given list of processes to find the processes having maximum burst time and initialize waiting time of each process with zero.
    a. If number of processes are odd then
    b. Take burst time of the middle process and assign this value to the time quantum
4. Else

a. Take the average of burst time of two middle most processes and assign this value to the time quantum

5. Find maximum number of time each process will execute by dividing maximum burst time with time quantum then add one in the result.
6. Initialize an array with zero that is used for storing the burst time that has been completed.
7. Iterate through the given list of processes.
    a. Initialize a variable with zero that is used as a counter.
    b. Iterate until the burst time of the process is greater than zero.
    c. If burst time is greater than or equal to time quantum then
        i. Store remaining burst time
        ii. Store completed burst time
        iii. Increment counter
    d. Else
        i. Store completed burst time
        ii. Assign zero to burst time variable
        iii. Increment counter
    e. Assign value of counter minus one in counter array
8. Iterate through the list of processes
    a. Iterate through the length of counter array
        i. If value of variable used for counter is equal to the counter of processes then
            1. Iterate through the process coming from the list of processes
            2. If value of that process is not equal to the upcoming process then

a. Add the waiting time of the upcoming process with the burst time completed.

ii. Else

1. Iterate through the list of processes
2. If that process is not equal to the upcoming process then
    a. Add waiting time of the upcoming process with the burst time completed.

9. Iterate through the list of processes
    a. Add total waiting time with waiting time of each process to find total waiting time
    b. Add burst time and waiting time of each process to find turnaround time
    c. Add total turnaround time and turnaround time of each process to find total turnaround time
10. Average waiting time is calculated by diving total waiting time with total number of processes.
11. Average turnaround time is calculated by dividing total turnaround time with total number of processes.

3.4 Pseudo Code of SJRR Scheduling Algorithm

```
burst ← 0
max ← 0
temp ← 0
total_tatime ← 0.0
tw_time ← 0.0
avg_wt ← 0.0
avg_tatime ←0.0

For I ← process-1 to 0
  For j ← 1 to process
    IF btime[j-1] > btime[j]
        temp ← btime[j-1]
        btime[j-1] ← btime[j]
        btime[j] ← temp
        ptemp ← proname[j-1]
        proname[j-1] ← proname [j]
        proname[j] ← ptemp

For I ← 0 to process
   btime[i] ← bu[i]
```

```
        proname[i] ← pname[i]
        IF max < btime[i]
            max ← btime[i]
            b[i] ← btime[i
            wtime[i] ← 0

    IF process%2!=0
        mid← (process+1)/2
    t← btime[mid]
    ELSE
        mid ← process/2
    t← (btime[mid]+btime[mid+1])/2

    dim ← max/t + 1

    For I ←0 to process
      For j ← 0 to dim
         r [I,j] ← 0

    For I ← 0 to process
      j ←0
      While btime[i]>0
          IF btime[i] >=t
              btime[i] ←btime[i]-t
              r[I,j]  ← t
              j ← j+1
           ELSE
               r[I,j] ← btime[i]
               btime[i] ← 0
               j←j+1
    counter[i] ← j-1


For j ←0  to process
    For I ← 0 to counter [j]
      IF I = = counter[j]
          For k← 0 to j
            IF k != j
                wtime[j] ← wtime[j] + r[k,i]
      ELSE
          For k ←o to process
            IF k != j
                wtime[j] ← wtime[j] +r[k, i]

      For j ←0 to process
        tw_time ← tw_time + wtime[j]
        tatime[j] ← b[j] + wtime[j]
        total_ tatime ← total_tatime+ tatime[j]
```

avg_wt ← tw_time / process
avg_tatime ← total_tatime/ process

3.5 Flow Chart of SJRR Scheduling Algorithm

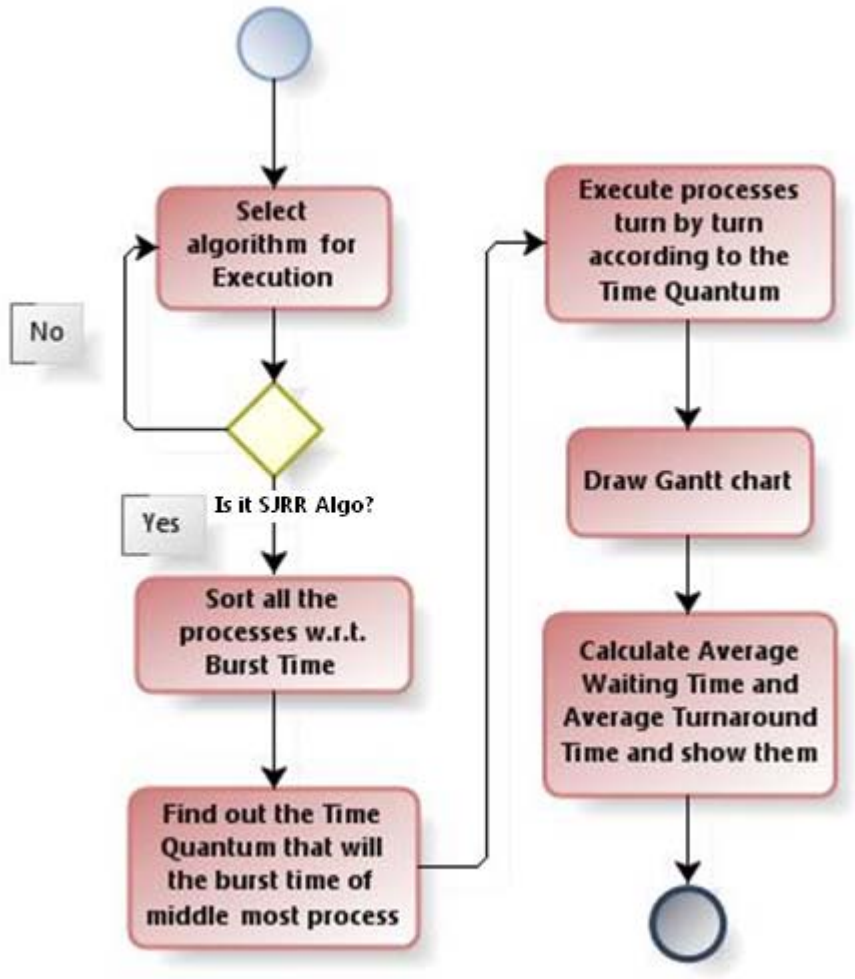


Figure 15: Flow Chart of SJRR CPU Scheduling Algorithm using Simulator

3.6 Example of SJRR Scheduling Algorithm

Assume the following set of processes with their burst time and arrival time:

Process Name	P1	P2	P3	P4	P5
Burst Time	10	29	3	7	12
Arrival Time	0	1	2	3	4

Original List

10	29	3	7	12

- Process having maximum burst time is P2 and maximum burst time i.e. max=29
- Burst time of each process;

  b[1] = 10,
  b[2] = 29,
  b[3] = 3,
  b[4] = 7,
  b[5] = 12
- Waiting time of each process will be zero at start

  wtime[1] = 0,
  wtime[2] = 0,
  wtime[3] = 0,
  wtime[4] = 0,
  wtime[5] = 0

- Total Number of processes, count= 5
- I = 4 then total passes in which processes will sort in ascending order are 4.
- As j =1; control will be at first location in given list. Here the burst time is 10; which is compared with burst time at second location. As 10<29, so in the next list 10 and 29 will be placed at the prior locations.

Original List

10	29	3	7	12

New List

10	29	3	7	12

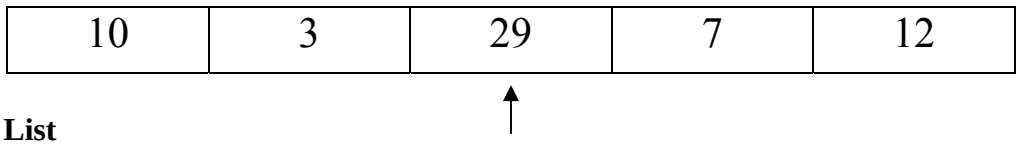
- As j=2; control will be at second location in given list. The burst time here is 29; which is compared with burst time at third location. As 29>3, therefore their places will be interchanged. Now in the next list 3 will be at second location and 29 will be at third location.

10	29	3	7	12

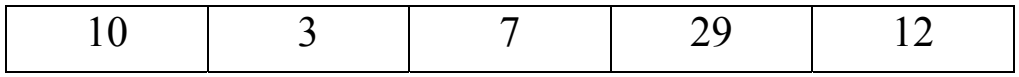
New List

10	3	29	7	12

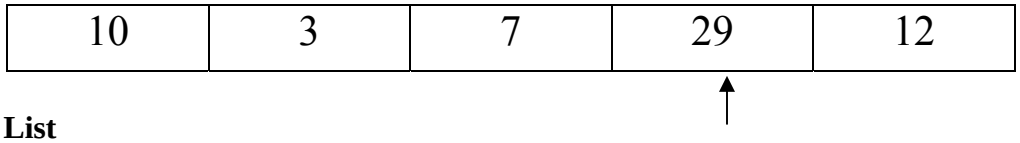
- As j=3; control will be at third location in given list. Here the burst time is 29; which is compared with burst time at fourth location. As 29>7, therefore their places will be interchanged. Now in the next list 7 will be at third location and 29 will be at fourth location.



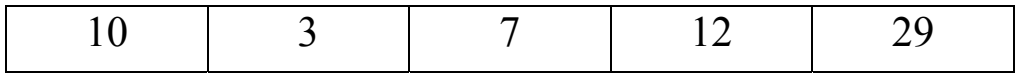
New List



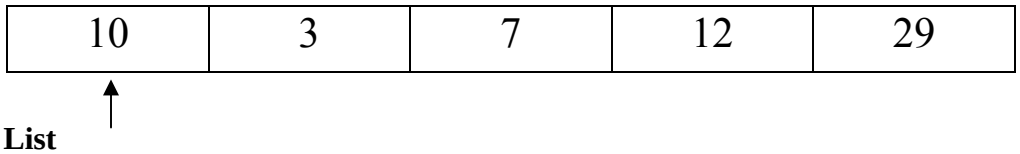
- As j=4; control will be at fourth location in given list. Here the burst time is 29; which is compared with burst time at fourth location. As 29>12, therefore their places will be interchanged. Now in the next list 12 will be at fourth location and 29 will be at fifth location



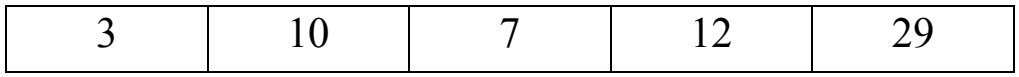
New List



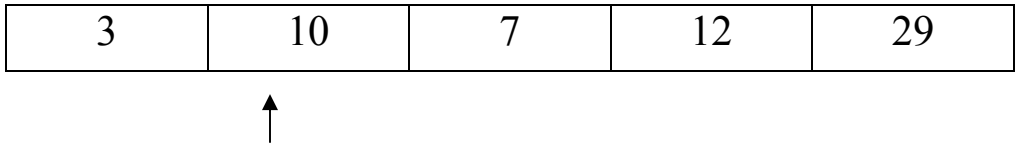
- Now I = 3, again iteration will start for j loop
- As j =1; control will be at first location in given list. Here the burst time is 10; which is compared with burst time at second location. As 10>3, therefore their places will be interchanged. Now in the next list 3 will be at first location and 10 will be at second location.



New List



- As j =2; control will be at second location in given list. Here the burst time is 10; which is compared with burst time at third location. As 10>7, therefore their places will be interchanged. Now in the next list 7 will be at second location and 10 will be at third location



New List

3	7	10	12	29

- As j =3; control will be at third location in given list. Here the burst time is 10; which is compared with burst time at fourth location. As 10<12, so in the next list 10 and 12 will be placed at the prior locations.

3	7	10	12	29

New List

3	7	10	12	29

- Now I = 2, again iteration will start for j loop and check the values of first three locations whether they are in ascending order or not. If they are in the ascending order then they remain left otherwise arrange them in that order.

- Now I = 1, again iteration will start for j loop and check the values of first two locations whether they are in ascending order or not. If they are in the ascending order then they remain left otherwise arrange them in that order.

- The **resultant or final list** will be

3	**7**	**10**	**12**	**29**

- According to the burst time arrangement, process name and their arrival times are also arranged.

Process Name	**P3**	**P4**	**P1**	**P5**	**P2**
Burst Time	3	7	10	12	29
Arrival Time	2	3	0	4	1

Sorting is completed here. Next step is to find out the time quantum that is used to execute processes.

- As number of processes i.e. count=5, count1 will be (5+1)/2 = 3
- Time quantum, t = btime[3-1] = btime[2] = 10

- Next step to calculate the maximum number of time longer burst time process will execute.

Dim = (29/10) +1 = 3.9 =3

- Next step is to allocate '0' value to arrays used to save the executed time of the processes.

  R_robin[0,0] = 0
  r_robin[0,1] = 0
  r_robin[0,2] = 0
  r_robin[1,0] = 0
  r_robin[1,1] = 0
  r_robin[1,2] = 0
  r_robin[2,0] = 0
  r_robin[2,1] = 0
  r_robin[2,2] = 0
  r_robin[3,0] = 0
  r_robin[3,1] = 0
  r_robin[3,2] = 0

- Next find executed time, remaining burst time of processes and value of counter that counts how many times a process will execute.
  - For first process,
    r_robin[0,0] = 3, btime[0] = 0 , counter[0] = 0
  - For second process,
    r_robin[1,0] = 7, btime[1] = 0 , counter[1] = 0
  - For third process,
    r_robin[2,0] = 10, btime[2] = 0 , counter[2] = 0
  - For fourth process,
    r_robin[3,0] = 10, btime[3] = 2
    r_robin[3,1] = 2, btime[3] = 0 , counter[3] = 1
  - For fifth process,
    r_robin[4,0] = 10, btime[4] = 19
    r_robin[4,1] = 10, btime[4] = 9
    r_robin[4,2] = 9, btime[4] = 0, counter[4] = 2

- Second last step is to find waiting time of each process;
  Waiting Time of P3, wtime[0] = 0
  Waiting Time of P4, wtime[2] = 3
  Waiting Time of P1, wtime[3] = 10
  Waiting Time of P5, wtime[4] = 30
  Waiting Time of P2, wtime[5] = 32

- Last step is to calculate total waiting time, average waiting time and average turnaround time;
  Total Waiting Time (tw_time) = 75

Average Waiting Time (avg_wt) = 15
Average Turnaround Time (avg_tatime) = 27.2

Gantt chart of above example

0 3	10	20	30	40	42	52	61
P3	**P4**	**P1**	**P5**	**P2**	**P5**	**P2**	**P2**

Graphical Representation of SJRR Gantt chart

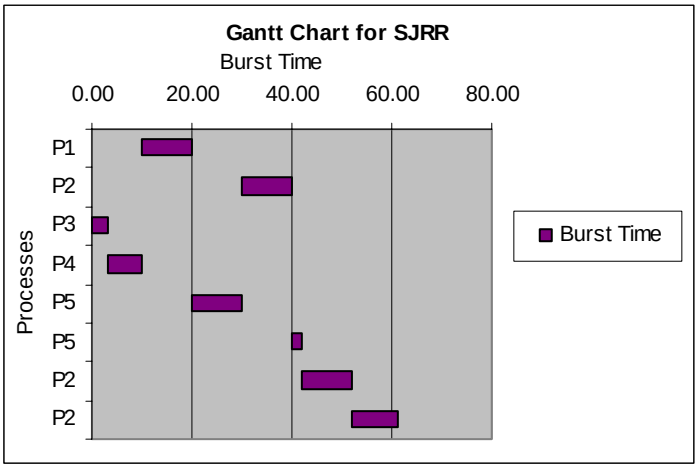


Figure 16: Graphical Representation of SJRR Gantt chart

3.7 C# Code of SJRR Scheduling Algorithm

To compare and contrast the working and performance of proposed SJRR CPU scheduling algorithm with the existing scheduling algorithms, we implemented these algorithms. Front end is implemented in C# and at back end Microsoft Access Database is used. C# code of proposed SJRR algorithm is presented here to make it more convenient.

```
Public void SJRR()
  {
      //variables used
      int I, j, k, burst = 0;
      int[] btime = new int[count];
      int[] b = new int[count];
      int[,] r_robin = new int[1000, 1000];
      int[] counter = new int[count];
      int max = 0, temp = 0;
      int dim;
      double total_tatime = 0.0, tw_time = 0.0, avg_wt = 0.0;
      double[] wtime = new double[count];
      double[] tatime = new double[count];
      String[] proname = new String[count];
      String ptemp = "";
```

```
//find maximum burst time
for (I = 0; I < count; i++)
{
   if (max < btime[i])
      max = btime[i];
   b[i] = btime[i];
   wtime[i] = 0;
}

//swap processes
for (I = count – 1; I > 0; i--)
{
   for (j = 1; j < count; j++)
   {
      if (btime[j – 1] > btime[j])
      {
         temp = btime[j – 1];
         btime[j – 1] = btime[j];
         btime[j] = temp;
         ptemp = proname[j – 1];
         proname[j – 1] = proname[j];
         proname[j] = ptemp;
      }
   }
}

//find time quantum
int count1 = 0, t=0;
if (count % 2 == 0)
{
   count1 = count / 2;
    t = (btime[count1-1] + btime[count1]) / 2;
}
else
{
   count1 = (count + 1) / 2;
   t = btime[count1-1];
}
MessageBox.Show("" + t);

//find maximum number of time,time quantum will be
//assigned
dim = max /t + 1;

//initializing Round robin array
for (I = 0; I < count; i++)
{
   for (j = 0; j <= dim; j++)
   {
      r_robin[I, j] = 0;
   }
}

//allocating time quantum and placing value in the
            //Rrobin array
I = 0;
while (I < count)
```

```
{
   j = 0;
   while (btime[i] > 0)
   {
      if (btime[i] >= t)
      {
         btime[i] = btime[i] – t;
         r_robin[I, j] = t;
         j++;
      }
      else
      {
         r_robin[I, j] = btime[i];
         btime[i] = 0;
         j++;
      }
   }
   counter[i] = j – 1;
   i++;
}

//find waiting time of each process
for (j = 0; j < count; j++)
{
   for (I = 0; I <= counter[j]; i++)
   {
      if (I == counter[j])
      {
         for (k = 0; k < j; k++)
         {
            if (k != j)

               wtime[j] += r_robin[k, i];
         }
      }
      else
         for (k = 0; k < count; k++)
         {
            if (k != j)
               wtime[j] += r_robin[k, i];

         }
   }

}

//calculating total waiting time, average waiting time
            //and total turnaround
for (j = 0; j < count; j++)
{
   tw_time = tw_time + wtime[j];
   tatime[j] = b[j] + wtime[j];
   total_tatime += tatime[j];

}
avg_wt = tw_time / count;
total_tatime = total_tatime / count;
```

```
    //catch exception
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
}
```

3.8 Steps of Combinatory Scheduling Algorithm

1. Take list of processes, their burst time and arrival time.
2. Find the factor F by adding burst time and arrival time of processes.
3. On the basis of factor, arrange processes and their relative burst time in ascending order using any sorting technique.
4. Calculate waiting time of each process.
5. Iterate through the list of processes
    a. Add total waiting time with waiting time of each process to find total waiting time
    b. Add burst time and waiting time of each process to find turnaround time
    c. Add total turnaround time and turnaround time of each process to find total turnaround time
6. Average waiting time is calculated by diving total waiting time with total number of processes.
7. Average turnaround time is calculated by dividing total turnaround time with total number of processes.

3.9 Pseudo Code of Combinatory Scheduling Algorithm

```
f ← 0
temp ← 0
total_tatime ← 0.0
tw_time ← 0.0
avg_wt ← 0.0
avg_tatime ←0.0

For I ← 0 to process
  factor[i] ← btime[i] + atime[i]

For I ← process-1 to 0
  For j ← 1 to process
    IF factor [j-1] > factor[j]
```

```
            f ← f actor[j-1]
            factor [j-1] ← factor[j]
            factor [j] ← f
            temp ← btime[j-1]
            btime[j-1] ← btime[j]
            btime[j] ← temp
            ptemp ← proname[j-1]
            proname[j-1] ← proname [j]
            proname[j] ← ptemp

wtime [1] ← 0
For j ← 1 to count
  wtime[j]  ← btime [j-1] + wtime [j-1]

For j ←0 to process
     tw_time ← tw_time + wtime[j]
     tatime[j] ← b[j] + wtime[j]
     total_ tatime ← total_tatime+ tatime[j]
avg_wt ← tw_time / process
avg_tatime ← total_tatime/ process
```

3.10 Flow Chart of Combinatory Scheduling Algorithm

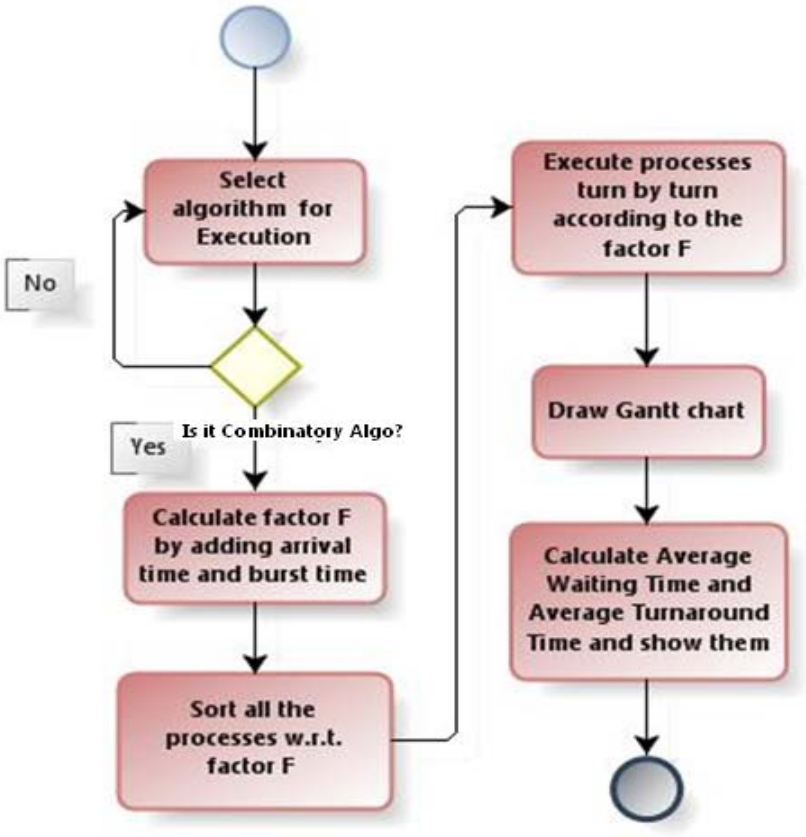


Figure 17: Flow Chart of Combinatory CPU Scheduling Algorithm using Simulator

3.11 Example of Combinatory Scheduling Algorithm

Assume the following set of processes with their burst time, arrival time and factor:

Process Name	P1	P2	P3	P4	P5
Burst Time	10	29	3	7	12
Arrival Time	0	1	2	3	4
Factor	10	30	5	10	16

Factor (i.e. addition of Arrival Time and Burst Time) of Processes used for sorting;

10	30	5	10	16

- Total Number of processes, count= 5
- I = 4 then total passes in which processes will sort in ascending order are 4.
- As j =1; control will be at first location in given list. Here the factor is 10; which is compared with factor at second location. As 10<30, so in the next list 10 and 29 will be placed at the prior locations.

Original List

10	30	5	10	16

New List

10	30	5	10	16

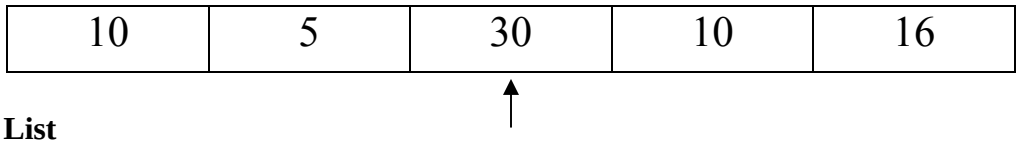
- As j=2; control will be at second location in given list. The factor here is 30; which is compared with factor at third location. As 30>5, therefore their places will be interchanged. Now in the next list 5 will be at second location and 30 will be at third location.

10	30	5	10	16

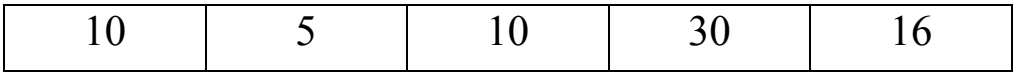
New List

10	5	30	10	16

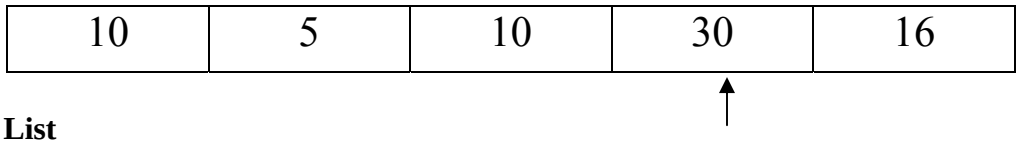
- As j=3; control will be at third location in given list. Here the factor is 30; which is compared with factor at fourth location. As 30>10, therefore their places will be interchanged. Now in the next list 10 will be at third location and 30 will be at fourth location.



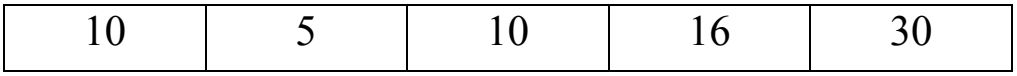
New List



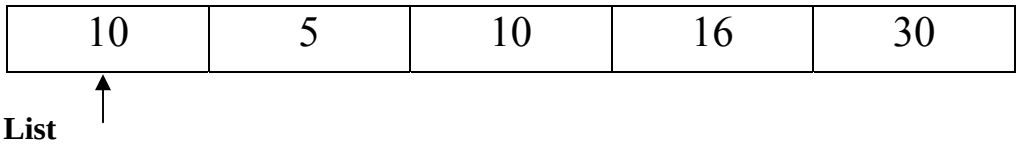
- As j=4; control will be at fourth location in given list. Here the factor is 30; which is compared with factor at fifth location. As 30>16, therefore their places will be interchanged. Now in the next list 16 will be at fourth location and 30 will be at fifth location



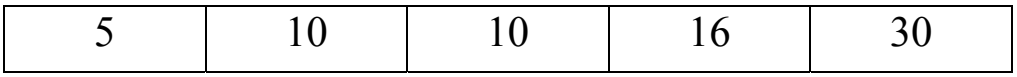
New List



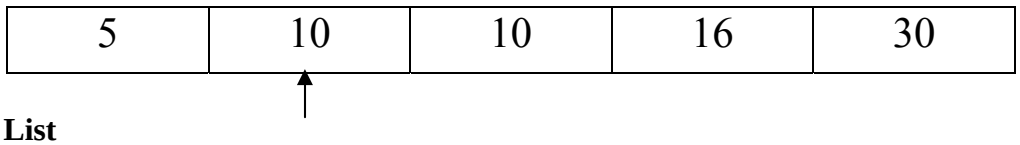
- Now I = 3, again iteration will start for j loop
- As j =1; control will be at first location in given list. Here the factor is 10; which is compared with factor at second location. As 10>5, therefore their places will be interchanged. Now in the next list 5 will be at first location and 10 will be at second location.



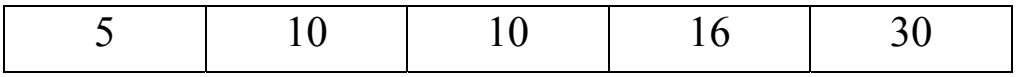
New List



- As j =2; control will be at second location in given list. Here the factor is 10; which is compared with factor at third location. As 10=10, therefore their places will not be interchanged.

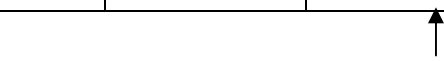


New List



- As j =3; control will be at third location in given list. Here the factor is 10; which is compared with factor at fourth location. As 10<16, so in the next list 10 and 16 will be placed at the prior locations.

5	10	10	16	30



New List

5	10	10	16	30

- Now I = 2, again iteration will start for j loop and check values of first three locations whether they are in ascending order or not. If they are in ascending order then they remain left otherwise arrange them in that order.

- Now I = 1, again iteration will start for j loop and check values of first two locations whether they are in ascending order or not. If they are in ascending order then they remain left otherwise arrange them in that order.

- The **resultant or final list** will be

5	**10**	**10**	**16**	**30**

- According to the factor's arrangement, process name and their burst times are also arranged.

Process Name	**P3**	**P1**	**P4**	**P5**	**P2**
Burst Time	3	10	7	12	29
Factor	5	10	10	16	30

Sorting is completed here. Next step is to find the waiting time of each process.

- Find waiting time of each process;

  Waiting time of P3, wtime[0] = 0
  Waiting time of P1, wtime[1] = 3
  Waiting time of P4, wtime[2] = 13
  Waiting time of P5, wtime[3] = 20
  Waiting time of P2, wtime[4] = 32

- Last step is to calculate total waiting time, average waiting time and average turnaround time;

  Total Waiting Time (tw_time) = 68
  Average Waiting Time (avg_wt) = 13.6
  Average Turnaround Time (avg_tatime) = 25.8

Gantt chart of above example

0 3	13	20	32	61
P3	**P1**	**P4**	**P5**	**P2**

Graphical Representation of Combinatory Gantt chart

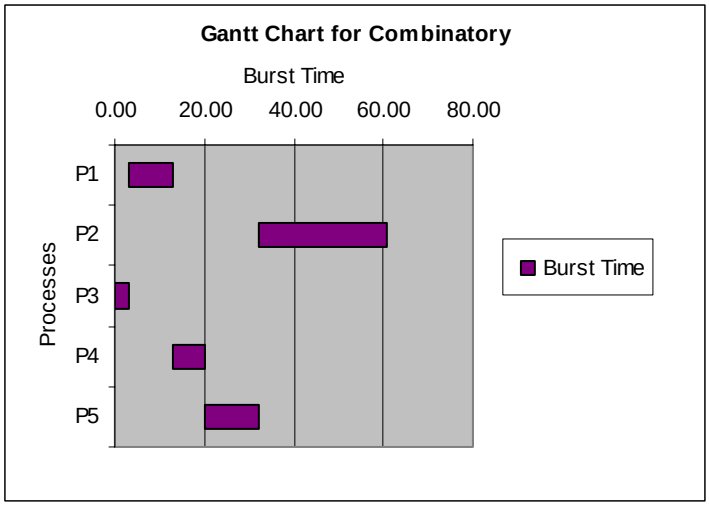


Figure 18: Graphical Representation of Combinatory Gantt chart

3.12 C# Code of Combinatory Scheduling Algorithm

```
public void Combinatory()
    {
        //variables used
        int I, j, k;
        int[] btime = new int[count];
        int[] atime = new int[count];
        int[] factor = new int[count];
        int temp = 0, f = 0;
        double total_tatime = 0.0, tw_time = 0.0, avg_wt = 0.0;
        double[] wtime = new double[count];
        double[] tatime = new double[count];
        String[] proname = new String[count];
        String ptemp = "";

        //find factor
        for (I = 0; I < count; i++)
        {
           factor[i] = btime[i] + atime[i];

        }
        //swap processes
        for (I = count – 1; I > 0; i--)
        {
           for (j = 1; j < count; j++)
           {
```

```
            if (factor[j – 1] > factor[j])
            {
               f = factor[j – 1];
               factor[j – 1] = factor[j];
               factor[j] = f;
               temp = btime[j – 1];
               btime[j – 1] = btime[j];
               btime[j] = temp;
               ptemp = proname[j – 1];
               proname[j – 1] = proname[j];
               proname[j] = ptemp;
            }

         }
      }
      //calculate waiting time
      wtime[1] = 0;
      for (j = 1; j < count; j++)
      {
         wtime[j] = btime[j – 1] + wtime[j – 1];

      }

                  //calculate total waiting time, average waiting time //and total turnaround
      for (j = 0; j < count; j++)
      {
         tw_time = tw_time + wtime[j];
         tatime[j] = btime[j] + wtime[j];
         total_tatime += tatime[j];
      }
      avg_wt = tw_time / count;
      total_tatime = total_tatime / count;
   }
   catch (Exception ex)
   {
      MessageBox.Show(ex.ToString());
   }
}
```

3.13 Performance Parameters

There are two orientations of performance criteria; these are user oriented criteria and system oriented criteria. Waiting time and turnaround time come under user oriented criteria and CPU utilization comes under system oriented criteria. Proposed algorithms has been implemented using C# and executed many times using different processes sets and it was analyzed that proposed SJRR algorithm gives minimum average waiting time and minimum average turnaround time as compared to other existing CPU scheduling algorithms. It also maximizes CPU utilization. Some other criteria like starvation, fairness, context switching were also tested. There is no problem of starvation in the proposed SJRR algorithm because it gives equal time unit to each process for execution.

It is appropriate for time sharing system because it provides fair share of CPU to each process. It does not incur much overhead of context switching as compared to round robin scheduling algorithm because in the proposed algorithm, time quantum is not generated randomly rather a specific formula is used to calculate it by keeping in mind the process burst time. Combinatory algorithm gives minimum average waiting time and minimum average turnaround time as compared to existing CPU scheduling algorithms as well as SJRR scheduling algorithm. In Combinatory scheduling algorithm, starvation has been removed to a great extent and there is no problem of context switching in it. In next chapter results and comparisons of new proposed algorithms are discussed.

3.14 Summary

The proposed SJRR CPU scheduling algorithm is a new algorithm which can be viewed as an improvement in round robin scheduling algorithm. It is a simple algorithm which uses concept of sorting to arrange the element in proper order then uses calculated time quantum to execute the processes. The proposed Combinatory CPU scheduling algorithm is also a new algorithm that uses a new factor F by adding two basic factors of scheduling algorithms (arrival time, burst time) and on the basis of that factor arranges the elements in ascending order in ready queue. After it, CPU is assigned one by one to each process. Performance parameters of both of these proposed algorithms have been discussed to observe the performance and efficiency of the new CPU scheduling algorithms. The strengths and weaknesses have also been discussed to present an overall picture. The implementation and experimental results of these algorithms will be presented in next chapter.

Chapter 4

4. IMPLEMENTATION AND RESULTS

To compare and contrast the performance of the proposed SJRR and Combinatory CPU scheduling algorithms, software named CPU Scheduling Algorithms Simulator was developed and used to check the performance of the proposed methodologies. In the forthcoming sections the details and results along with observations are presented.

4.1 Introduction

CPU Scheduling Algorithms Simulator was implemented using C#.Net and MS Access. Following are system requirements to run the developed software:

a. Windows XP or later
b. Dot Net Framework 2.0 or later
c. MS Access 2000 or later

All well known CPU scheduling algorithms have been implemented and are integrated in this software. List of the processes along with their arrival time, burst time and priority are entered by the users and saved into the database. After that users can select any of the two implemented algorithms. The software uses same list of processes their arrival time burst time and priority by both of the selected algorithms to schedule the processes. Scheduled list of processes is displayed in the form of Gantt chart along with the total waiting time, average waiting time and average turnaround time in milliseconds required to the processes for scheduling. Following sections give the detail description of the software.

4.2 Software Details

The CPU Scheduling Algorithms Simulator has been developed for detail study and evaluation of CPU scheduling algorithms. It is a comprehensive software tool that runs simulation, generates useful data for performance evaluation of algorithm and provides user friendly environment. A user friendly and mouse driven graphical user interface (GUI) is designed so that users can easily understand the environment and interact with

the system. The system is designed to simulate the selected algorithms. Block Diagram of this system is given below:

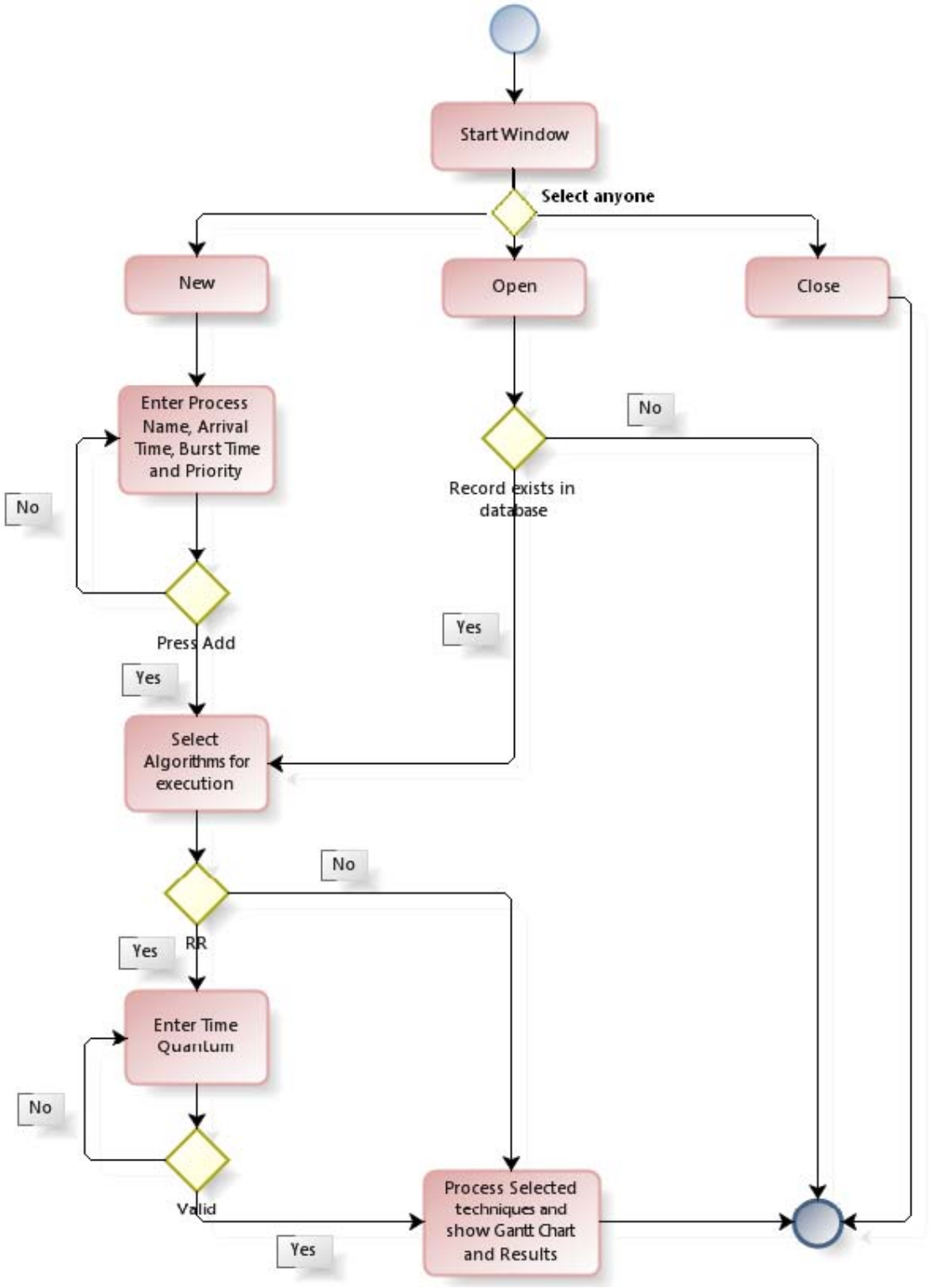


Figure 19: Block Diagram of CPU Scheduling Algorithm Simulator

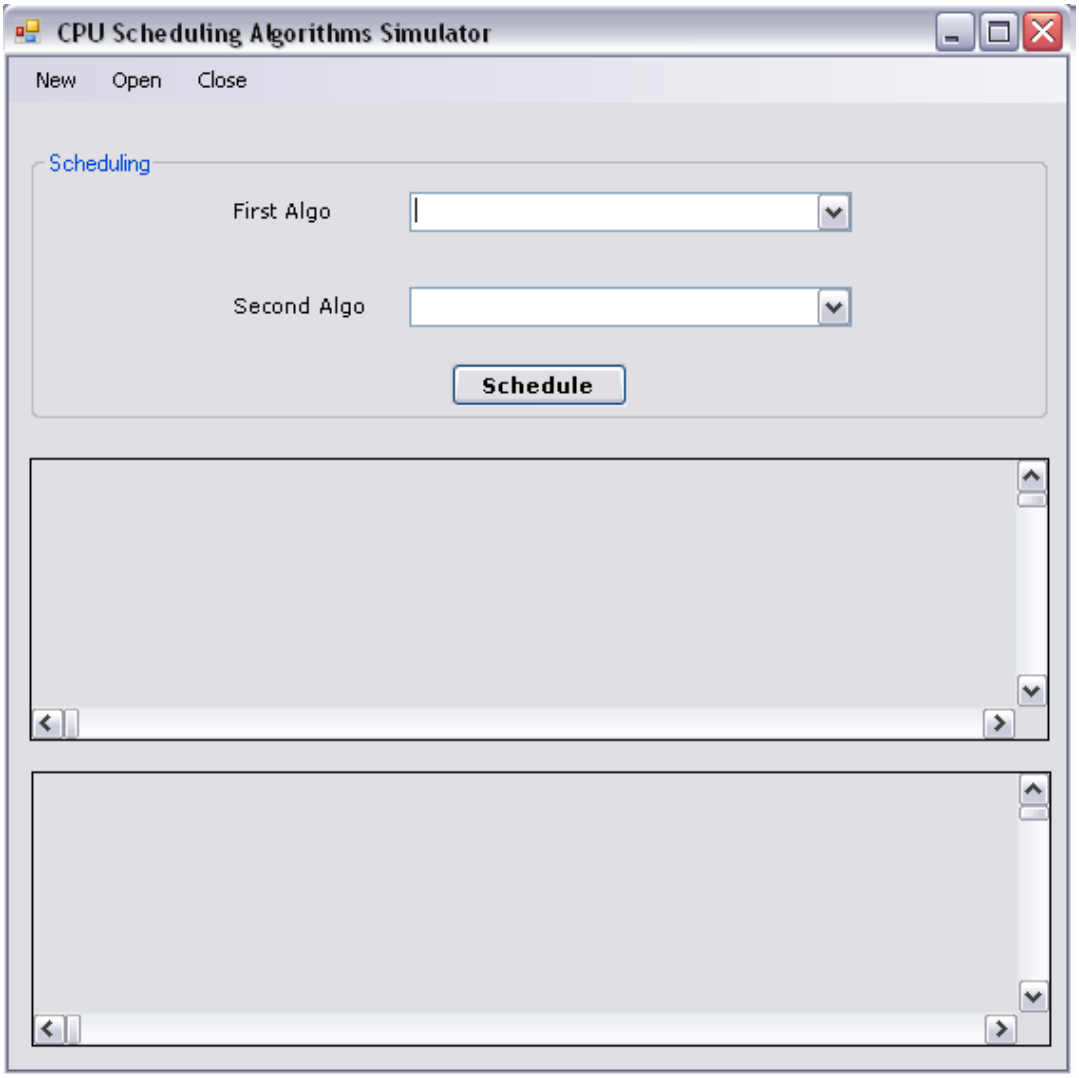


Figure 20: CPU Scheduling Algorithms Simulator

Figure 20 shows the main window of the CPU Scheduling Algorithms Simulator. Here, three menu items are shown that are New, Open and Close. When user clicks on 'New' menu item, a window named 'Add Process Information' will open that is used to enter all the input related to process. Process Information includes:

- Process Name
- Arrival time
- Burst Time
- Priority

Two menu items are present on this window. One is Add and other is Cancel. When user want to add the given information into the database, he/ she will click on 'Add' menu item, all the information will save into the database. If user does not want to add the information of the process into the database, he/ she simply clicks 'Cancel' menu item, textboxes will be empty and no record will save into the database.

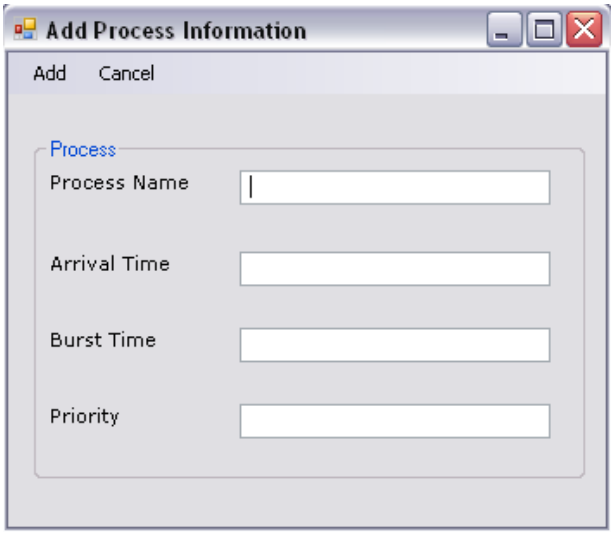
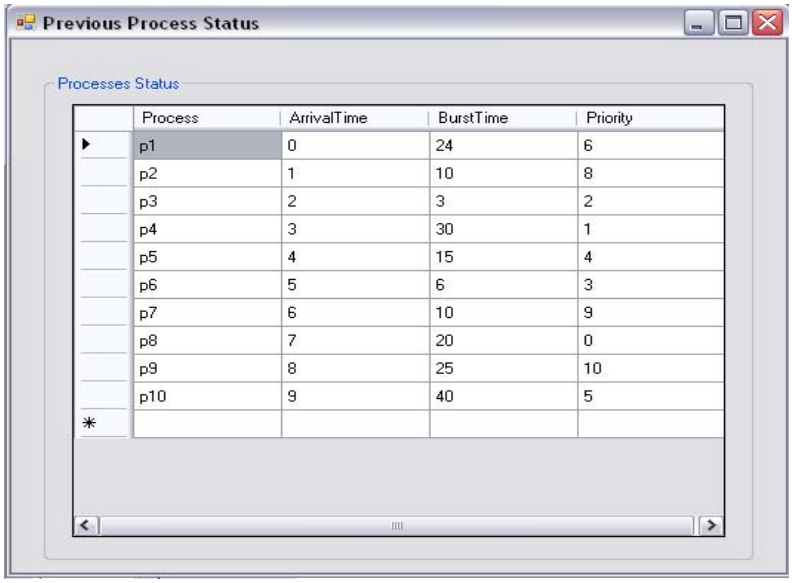


Figure 21: Add Process Information Window

When user clicks on Open menu item in main window i.e. CPU Scheduling Algorithms Simulator, a window named 'Previous Process Status' will open that contains all the record of the processes. Figure 22 shows Previous Process Status window.



Process	ArrivalTime	BurstTime	Priority
p1	0	24	6
p2	1	10	8
p3	2	3	2
p4	3	30	1
p5	4	15	4
p6	5	6	3
p7	6	10	9
p8	7	20	0
p9	8	25	10
p10	9	40	5

Figure 22: Previous Process Status

Third menu item in the main window is ‘Close’ that is used to close the main window. After entering new processes and their related information, user needs to select two algorithms by using two drop down lists from the main window. ‘Schedule’ button is used for scheduling the list of processes using two selected scheduling algorithms. Output of the scheduling is represented in the form of Gantt chart in the panels. First panel is used for the Gantt chart of the first selected algorithm and second panel is used for the Gantt chart of the second selected algorithm as shown in the Figure 23. In this figure comparison of two algorithms SJRR and Combinatory is displayed. If the list of processes is large then complete Gantt chart will not be displayed in both panels, users have to move the scroll bars to see the complete Gantt chart.

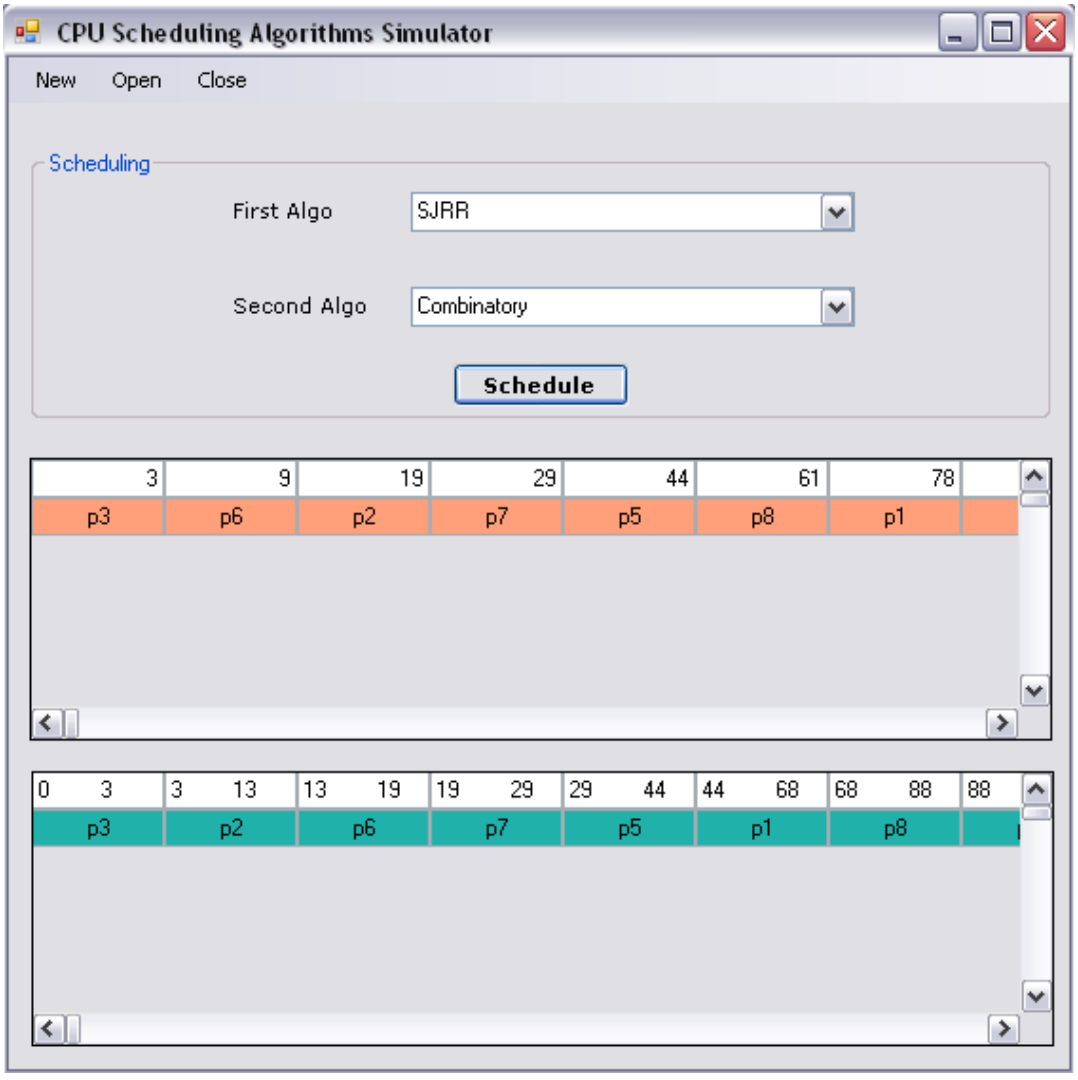


Figure 23: Gantt chart representation using SJRR and Combinatory Algorithms

The waiting time, average waiting time and average turnaround time of both selected algorithms are represented in milliseconds in two message boxes respectively.

Figure 24: Results of SJRR Algorithm

Figure 25: Results of Combinatory Algorithm

The CPU Scheduling Algorithms Simulator is designed only to evaluate the performance of new proposed scheduling algorithms and compare it with the performance of existing algorithms. Therefore no actual computing work that is essential in real multiprogramming environment is required in it.

4.3 Experiments:

Developed software was used and simulator was run to check the performance of the proposed SJRR and Combinatory CPU Scheduling Algorithms on a computer with following specifications

d. 1. GHz Intel Processor or later
e. 512 MB of RAM

4.4 Results:

New proposed CPU scheduling algorithms, SJRR and Combinatory algorithms are compared with all implemented scheduling algorithms. For each comparison, different number of processes along with their arrival time, burst time and priority are taken and then they are scheduled. The numbers of processes in each list were 5, 10, 15, 20, 25, 30, 50, 100 and 200. In the upcoming section results of these experiments in the form of graphs as well as textual description is given

4.4.1 Comparison of SJRR and Combinatory Algorithms with First Come First Served (FCFS) Algorithm

As discussed earlier, First Come First Served (FCFS) scheduling algorithm is the simplest technique used so far. Therefore, we start our comparison with FCFS. Here in Table 1, average waiting time of FCFS, SJRR and Combinatory algorithms are taken for comparison.

No of Processes	5	10	15	20	25	30	50	100	200
FCFS (ms)	28	69.1	260.47	193.2	3730.4	7850	1316	5470.96	10260.39
SJRR (ms)	15	68.2	266.87	184.55	2782.8	6122.5	1092.06	3138.04	8004.37
Combinatory (ms)	13.6	52.5	221.3	159.5	2096.4	4495	838.06	2443.1	6162.34

Table 1: Comparison of Average Waiting Time of FCFS with SJRR and Combinatory

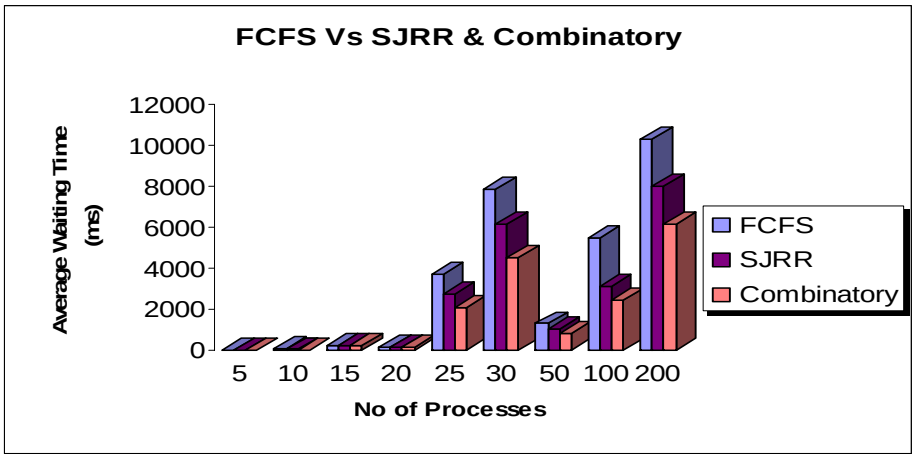


Figure 26: Graph FCFS Vs SJRR & Combinatory (Avg Waiting Time)

In Table 2, average turnaround time of FCFS and SJRR are taken for comparison.

No of Processes	5	10	15	20	25	30	50	100	200
FCFS (ms)	40.2	87.4	299.5	212.75	3992.4	8315	1366.9	5582.63	10366.97
SJRR (ms)	27.2	86.5	305.9	204.1	3044.8	6587.5	1142.76	3249.71	8109.95
Combinatory (ms)	25.8	70.8	260.4	179.05	2358.4	4960	888.76	2554.77	6267.93

Table 2: Comparison of Average Turnaround Time of FCFS with SJRR and Combinatory

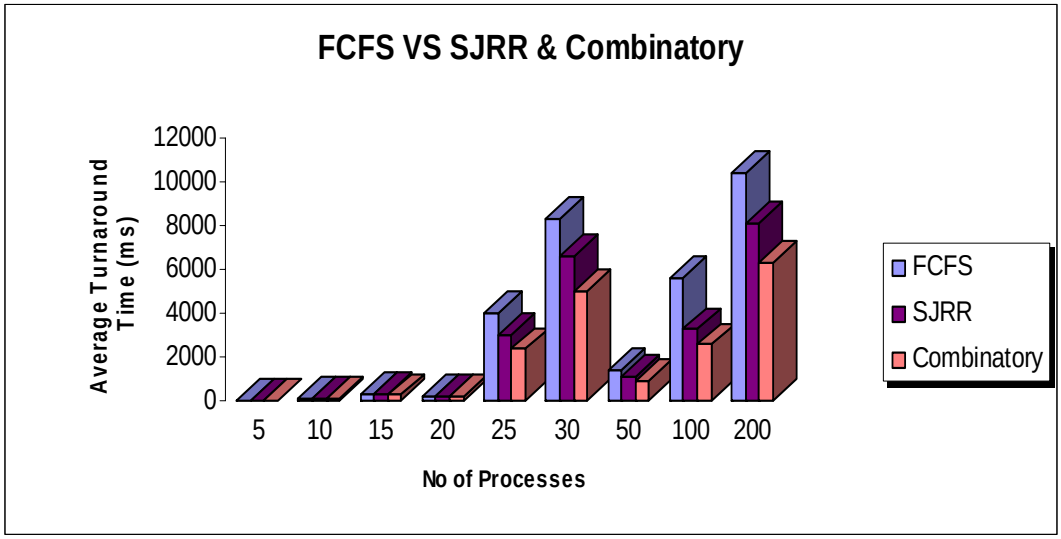


Figure 27: Graph FCFS Vs SJRR & Combinatory (Avg Turnaround Time)

In above two graphs of Figure 26 and 27, at x- axis number of processes are placed that we want to schedule. At y-axis of the first graph average waiting time of FCFS, SJRR and Combinatory in milliseconds is taken and of the second graph average turnaround time of FCFS, SJRR and Combinatory in milliseconds is taken. It is depicted from the above two graphs that the difference between average waiting time and average turnaround time of FCFS and SJRR is not more clear from 5 to 20 processes but it become quite large from up to 20 processes which makes SJRR more efficient than FCFS. The difference between average waiting time and average turnaround time of FCFS and Combinatory is quite large which makes Combinatory Algorithm much more efficient as compared to FCFS that is clear from above graphs.

4.4.2 Comparison of SJRR and Combinatory Algorithms with Shortest Job First (SJF) Algorithm

In earlier discussion it has been seen that SJF is an optimal CPU scheduling algorithm. Here we compare the performance of SJF with the performance of SJRR and Combinatory Algorithms. In Table 3, we take average waiting time of the SJF, SJRR and Combinatory algorithms and in Figure 28, graph show the comparison of the average waiting time of the algorithms.

No of Processes	5	10	15	20	25	30	50	100	200
SJF (ms)	13	51.2	213.67	152.15	2096.4	4495	800.66	2180.86	5900.25
SJRR (ms)	15	68.2	266.87	184.55	2782.8	6122.5	1092.06	3138.04	8004.37
Combinatory (ms)	13.6	52.5	221.3	159.5	2096.4	4495	838.06	2443.1	6162.34

Table 3: Comparison of Average Waiting Time of SJF with SJRR and Combinatory

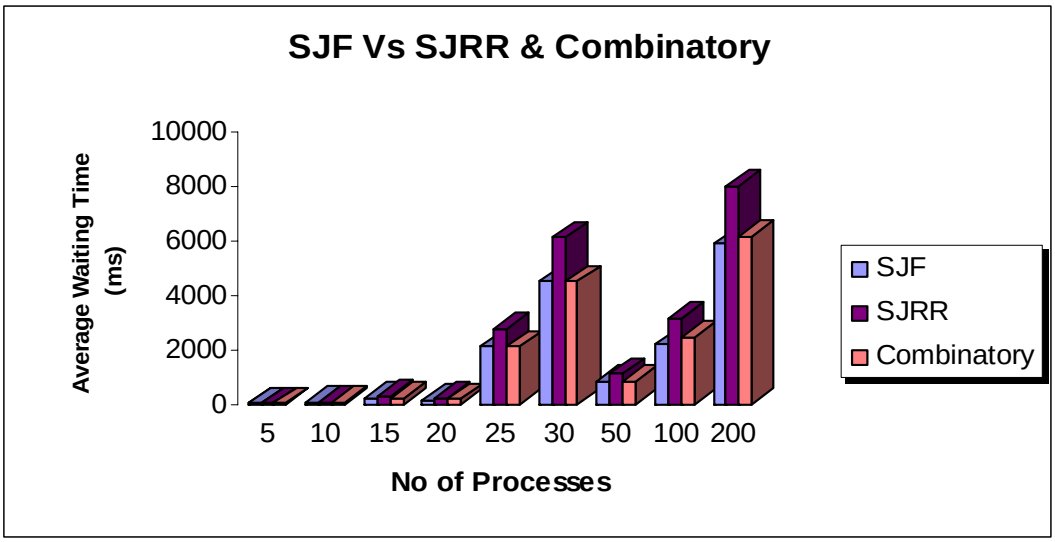


Figure 28: Graph SJF Vs SJRR & Combinatory (Avg Waiting Time)

In Table 4, we take average turnaround time of the SJF, SJRR and Combinatory algorithms and in Figure 29 graph show the comparison of the average turnaround time of the algorithms.

No of	5	10	15	20	25	30	50	100	200

Processes									
SJF (ms)	25.2	69.5	252.73	171.7	2358.4	4960	851.36	2292.53	6005.83
SJRR (ms)	27.2	86.5	305.9	204.1	3044.8	6587.5	1142.76	3249.71	8109.95
Combinatory (ms)	25.8	70.8	260.4	179.05	2358.4	4960	888.76	2554.77	6267.93

Table 4: Comparison of Average Turnaround Time of SJF with SJRR and Combinatory

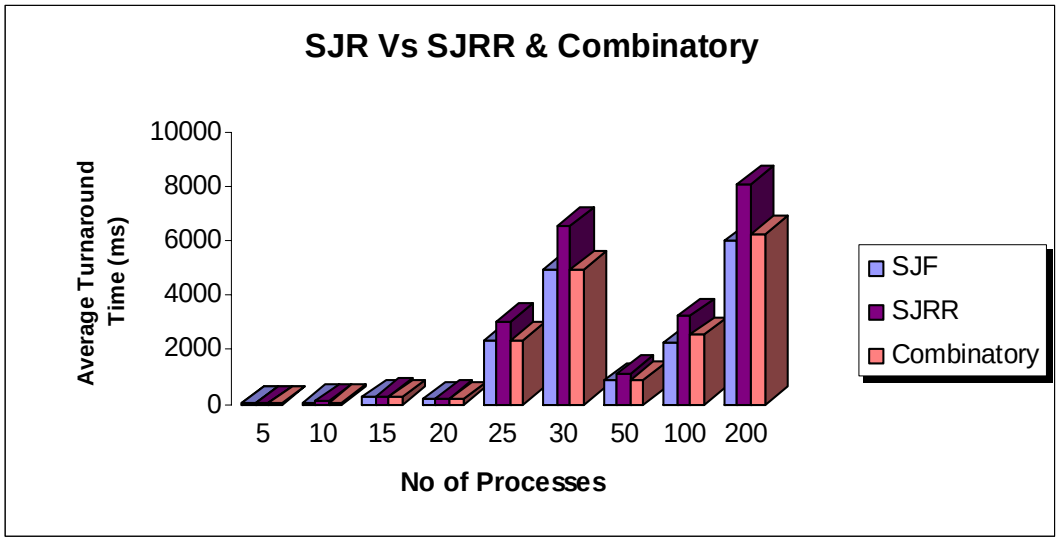


Figure 29: Graph SJF Vs SJRR & Combinatory (Avg Turnaround Time)

In above two graphs, at x-axis, we take number of processes, average waiting time at y-axis of graph of Figure 28 and average turnaround time at y-axis of graph of Figure 29. It has been cleared from the above data and graphs that SJRR shows small difference of average waiting time and average turnaround for small number of processes but this difference increase for large number of processes. Combinatory algorithm has almost same result like SJF. The difference between results of SJF and Combinatory algorithms is negligible for small number of processes but this difference slightly increases for larger list of processes. Combinatory algorithm also removes the overhead of starvation at much extent because it combines arrival time and burst time in the form of Factor F and on the basis of that factor arranges the processes in ascending order. But starvation is a great overhead in Shortest Job First (SJF) algorithm.

4.4.3 Comparison of SJRR and Combinatory Algorithms with Round Robin (RR) Algorithm

In this section, performance of Round Robin is compared with SJRR and Combinatory. Table 5 shows average waiting time of RR, SJRR and Combinatory algorithms and comparison is shown in the form of graph in Figure 30.

No of Processes	5	10	15	20	25	30	50	100	200
RR (ms)	23	86.2	332	242.5	4261.2	9051	1471.5	3787.76	8731.23
SJRR (ms)	15	68.2	266.87	184.55	2782.8	6122.5	1092.06	3138.04	8004.37
Combinatory (ms)	13.6	52.5	221.3	159.5	2096.4	4495	838.06	2443.1	6162.34

Table 5: Comparison of Average Waiting Time of RR with SJRR and Combinatory

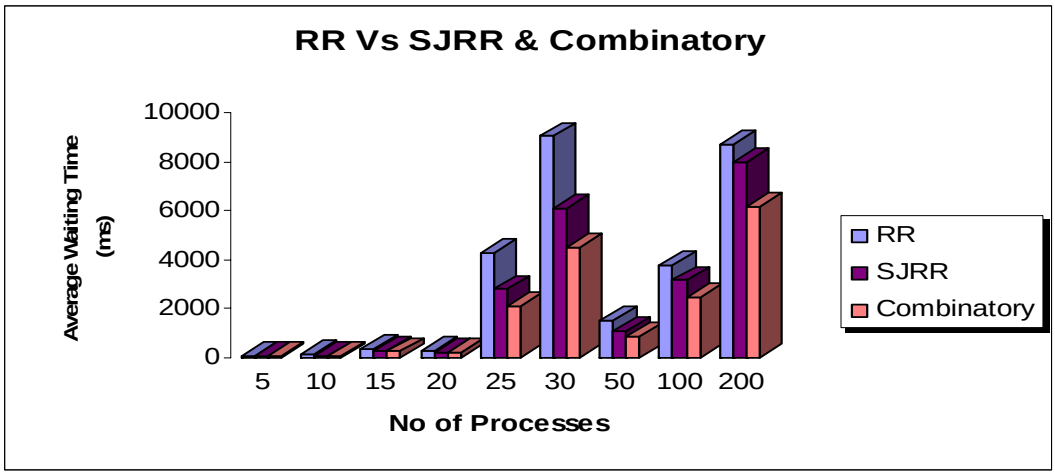


Figure 30: Graph RR Vs SJRR & Combinatory (Avg Waiting Time)

Table 6 shows average turnaround time of RR, SJRR and Combinatory algorithms and comparison is shown in the form of graph in Figure 31.

No of Processes	5	10	15	20	25	30	50	100	200
RR (ms)	35.2	104.5	371.07	242.5	4523.2	9051	1522.2	3899.43	8836.82
SJRR (ms)	27.2	86.5	305.9	204.1	3044.8	6587.5	1142.76	3249.71	8109.95
Combinatory (ms)	25.8	70.8	260.4	179.05	2358.4	4960	888.76	2554.77	6267.93

Table 6: Comparison of Average Turnaround Time of RR with SJRR and Combinatory

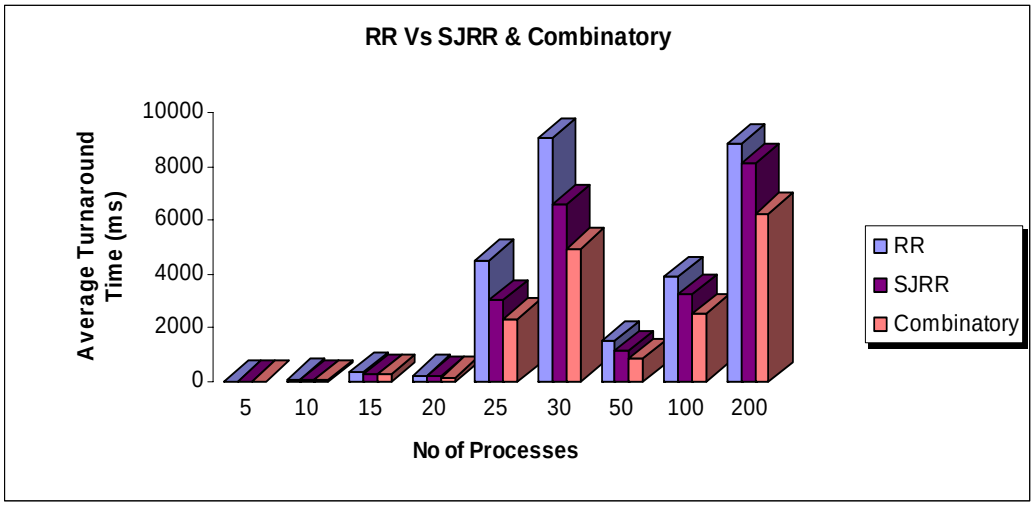


Figure 31: Graph RR Vs SJRR & Combinatory (Avg Turnaround Time)

At x-axis of both above graphs, no of processes are taken, at y-axis of the first graph, average waiting time and of the second graph average turnaround time is taken. It is obvious from the graphs that average waiting time and average turnaround time of Round Robin is greater than SJRR and Combinatory which makes both of these algorithms more efficient than Round Robin algorithm.

4.4.4 Comparison of SJRR and Combinatory Algorithms with Priority Algorithm

Now we compare the performance of priority algorithm with proposed algorithms. Following are the tables and graphs that show the relationship of average waiting time and average turnaround time of the priority algorithms with SJRR and Combinatory algorithms.

No of Processes	5	10	15	20	25	30	50	100	200
Priority (ms)	30.6	81.4	265.5	202.1	2991.6	6428	1314.7	5800.4	10567.89
SJRR (ms)	15	68.2	266.87	184.55	2782.8	6122.5	1092.06	3138.04	8004.37
Combinatory (ms)	13.6	52.5	221.3	159.5	2096.4	4495	838.06	2443.1	6162.34

Table 7: Comparison of Average Waiting Time of Priority with SJRR and Combinatory

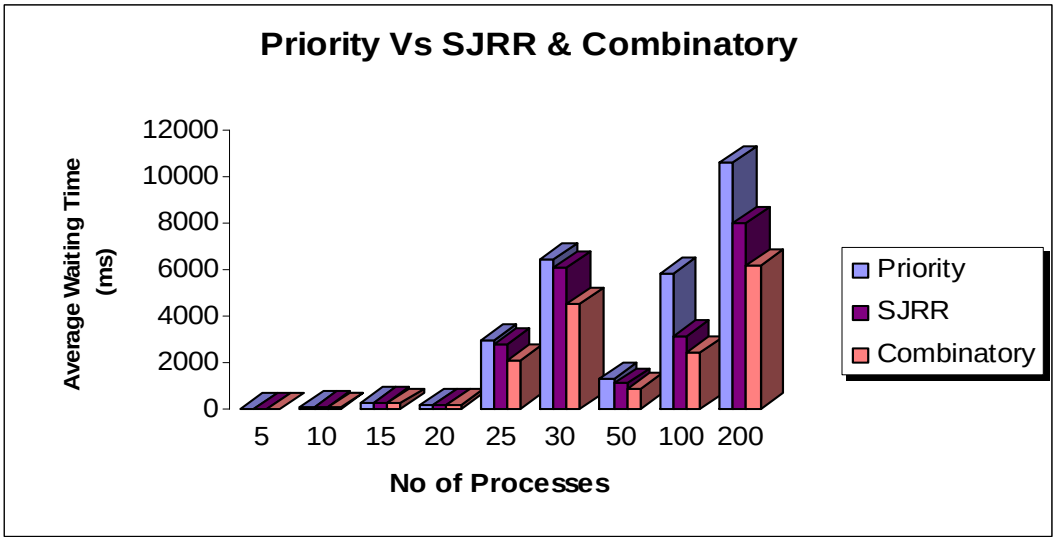


Figure 32: Graph Priority Vs SJRR & Combinatory (Avg Waiting Time)

No of Processes	5	10	15	20	25	30	50	100	200
Priority (ms)	42.8	99.7	304.6	221.6	3253.6	6893	1365.42	5912.07	10673.47
SJRR (ms)	27.2	86.5	305.9	204.1	3044.8	6587.5	1142.76	3249.71	8109.95
Combinatory (ms)	25.8	70.8	260.4	179.05	2358.4	4960	888.76	2554.77	6267.93

Table 8: Comparison of Average Turnaround Time of Priority with SJRR and Combinatory

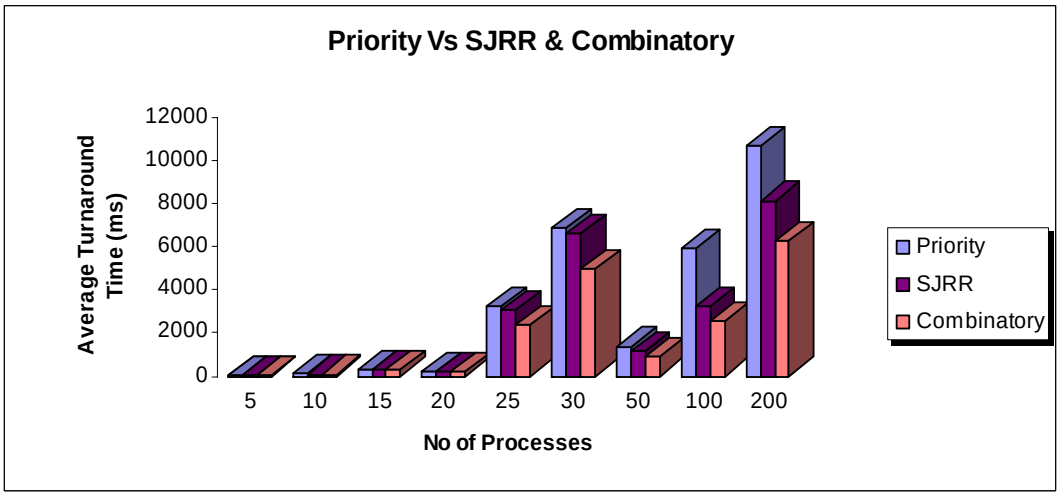


Figure 33: Graph Priority Vs SJRR & Combinatory (Avg Turnaround Time)

In Table 7 average waiting time of Priority, SJRR and Combinatory algorithms is taken and in Figure 32, no of processes are taken along x-axis and average waiting time is taken along y-axis. In Table 8 average turnaround waiting time of Priority, SJRR and

Combinatory algorithms is taken and in Figure 33, no of processes are taken along x-axis and average turnaround time is taken along y-axis. As shown in the graphs SJRR and Combinatory algorithms show better readings of average waiting time and average turnaround time as compared to Priority. There is a small difference between the results of average waiting time and average turnaround time of SJRR and the results of average waiting time and average turnaround time of Priority. This difference increases when we move towards large number of processes. This shows the performance of SJRR and Combinatory algorithms over Priority algorithm.

4.4.5 Comparison of SJRR and Combinatory Algorithms

In this section comparison of both the proposed algorithms SJRR and Combinatory is represented. Table 9 represents average waiting time of both of these algorithms and in Figure 34 graph is drawn to show the graphical comparison of these algorithms.

No of Processes	5	10	15	20	25	30	50	100	200
SJRR (ms)	15	68.2	266.87	184.55	2782.8	6122.5	1092.06	3138.04	8004.37
Combinatory (ms)	13.6	52.5	221.3	159.5	2096.4	4495	838.06	2443.1	6162.34

Table 9: Comparison of Average Waiting Time of SJRR and Combinatory

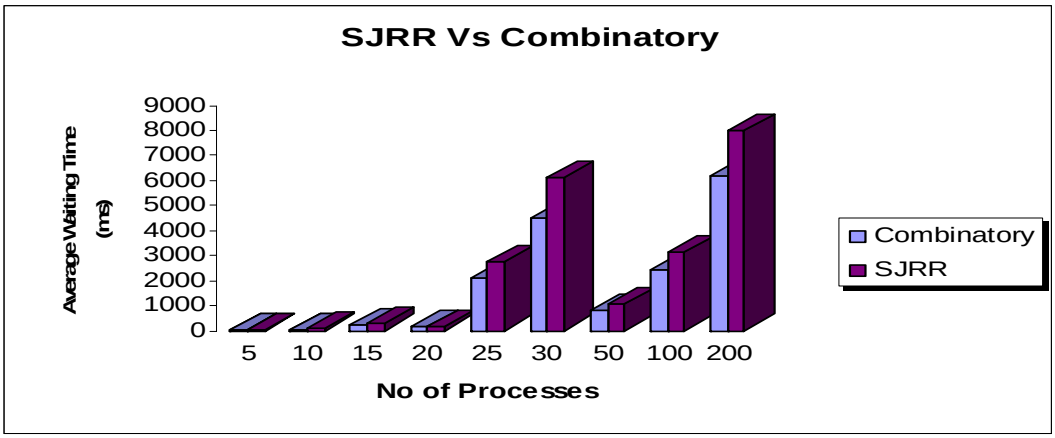


Figure 34: Graph SJRR Vs Combinatory (Avg Waiting Time)

Table 10 represents average turnaround time of both of these algorithms and in Figure 35 graph is drawn to show the graphical comparison of these.

No of Processes	5	10	15	20	25	30	50	100	200
SJRR (ms)	27.2	86.5	305.9	204.1	3044.8	6587.5	1142.76	3249.71	8109.95
Combinatory (ms)	25.8	70.8	260.4	179.05	2358.4	4960	888.76	2554.77	6267.93

Table 10: Comparison of Average Turnaround Time of SJRR and Combinatory

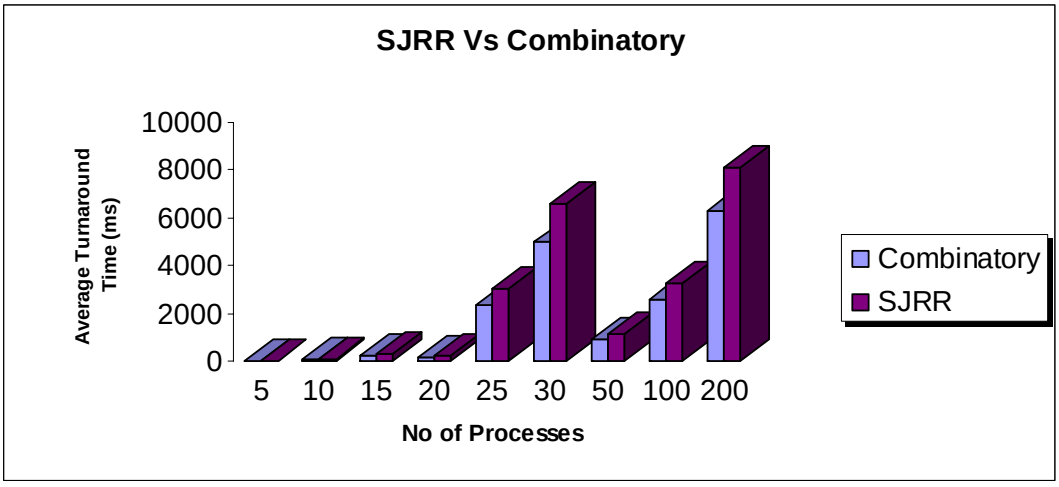


Figure 35: Graph SJRR Vs Combinatory (Avg Turnaround Time)

Along x-axis, different set of processes are taken for SJRR and Combinatory and along y-axis average waiting time and average turnaround time spend for scheduling these set of processes is taken. As shown from above graphs, Combinatory algorithm shows better reading of average waiting time and average turnaround time as compared to SJRR.

Below in the Figure 34 and 35 is the graphical description of cumulative comparison of average waiting time and average turnaround time of proposed SJRR and Combinatory Algorithms with FCFS, SJF, RR and Priority scheduling algorithms. Using these analysis results, it is concluded that SJRR shows marked improvement as compared to First Come First Server (FCFS), Round Robin (RR) and Priority algorithms. Combinatory algorithm also shows noticeable improvement as compared to FCFS, RR, Priority and SJRR. Along it, results of Combinatory algorithm is almost close to Shortest Job First (SJF).

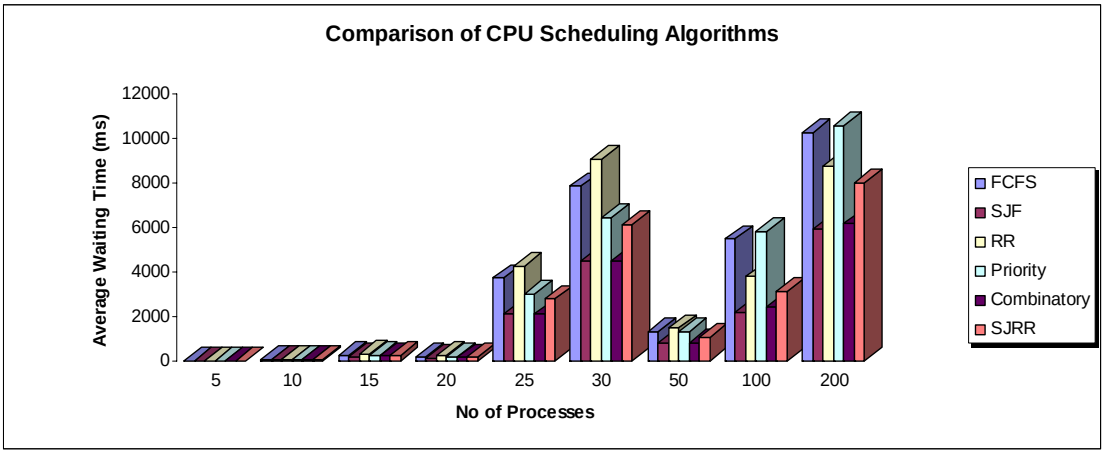


Figure 36: Cumulative comparison of Average Waiting Time of Scheduling Algorithms

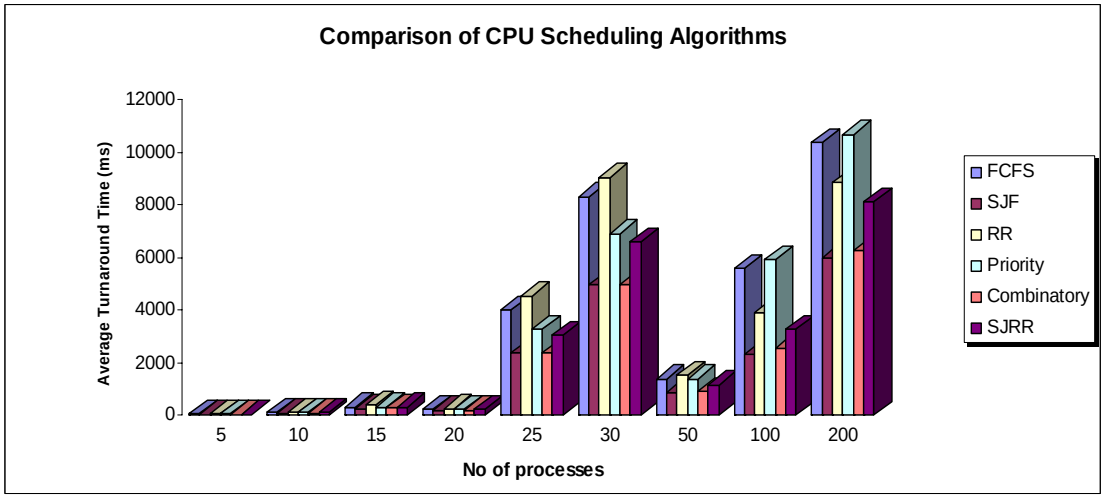


Figure 37: Cumulative comparison of Average Turnaround Time of Scheduling Algorithms

These experimental results have been used to derive performance metrics of scheduling algorithms. These metrics are given below in Table 11. This table gives detail information about performance of each scheduling algorithm.

	FCFS	*SJF*	*RR*	*Priority*	*SJRR*	*Combinatory*
Selection Function	min[a]	min[s]	constant	min[p]	constant	min[f]
Decision Mode	non pre-emptive	non pre-emptive	pre-emptive	non pre-emptive	RR	non pre-emptive
Turnaround Time	high	less	less for short processes	average	slightly higher than SJF	Almost equal to SJF
Waiting Time	high	less	less for short processes	average	slightly higher than SJF	Almost equal to SJF
CPU Utilization	less	less	High	less	high	higher than SJF
Starvation	no	possible	No	possible	no	possible but less than SJF

Overhead	minimal	can be high	Low	can be high	can be high	low
Context Switching	no	no	Yes	no	yes but less than RR	no
Time Sharing System	no	no	Yes	no	yes	no
Where a= arrival Time, s= total service time required by the process, p= priority number , f= burst time + arrival time						

Table 11: Performance Metrics of Scheduling Algorithms

4.5 Summary:

All of well known CPU Scheduling Algorithms were implemented with proposed SJRR and Combinatory algorithms to compare and contrast the performance of proposed CPU scheduling algorithms with existing CPU scheduling algorithms and derive performance metrics for all of these scheduling algorithms. These experiments were also made to check the position of proposed CPU scheduling algorithms in the entire community of CPU scheduling algorithms. In the next chapter conclusion and future work is presented in detail.

Chapter 5

5. CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In multiprogramming operating system, number of programs execute concurrently. The main objective of multiprogramming operating system is to share system resources among multiple users and system processes in order to maximize CPU utilization. In operating system functions, scheduling is an important one. Scheduling of resources plays a central role in efficient resource sharing. All computer resources are scheduled before use. CPU is the main computer resource; therefore its scheduling is vital.

CPU scheduling is a fundamental discipline which helps us to obtain clear understanding of complex set of policies and mechanisms used to govern the order in which tasks are executed by the processor. Different algorithms have been used to implement CPU scheduling. Each algorithm has its own properties that differentiate it from others.

Different evaluation methods have been used to evaluate various algorithms. These methods include Deterministic Modeling, Queuing Models and Simulations. Deterministic Modeling and Simulations are commonly used because they provide accurate evaluation of scheduling algorithms. A particular predetermined workload is taken in Deterministic Modeling that is used to determine performance of each algorithm. In Simulations, a model of the computer system is programmed.

In this work two new scheduling algorithms are proposed. One is for time sharing system and other is for multiprogramming system. The proposed algorithm that is used for time sharing system is SJRR scheduling algorithm. SJRR algorithm sorts all the incoming processes according to their burst time and then apply time quantum on these sorted processes to execute them. Time quantum is calculated by using formula not given by the user. The proposed algorithm that is used for multiprogramming system is Combinatory scheduling algorithm. In this algorithm a new factor F is attached with each incoming process that sums the effects of two basic factors (arrival time and burst time) of the process. The processes are arranged in ascending order on the basis of that factor and then CPU is assigned to each process for execution. The purpose of developing these

algorithms was to develop efficient algorithms that can overcome the limitations of existing algorithms. These proposed algorithms have achieved the aim at some level.

Implementation and results in Chapter 4 has shown the performance and position of the proposed algorithms. Results have shown that FCFS algorithm is simplest one that is easy to implement and have smaller computational overheads but its performance is poor, gives low throughput and long average waiting time and turnaround time. There is a problem of convoy effect in FCFS because small processes have to wait for longer processes to get off the CPU. Shortest Job First is an optimal scheduling algorithm that gives minimum average waiting time and turnaround time but there is problem of knowing the length of next CPU request. Round Robin algorithm is suitable for time sharing. It uses time quantum given by the user to execute the processes. Its performance completely depends on time quantum. If time quantum is large it gives results similar to FCFS but if time quantum is very small then there is a problem of context switching. In Priority scheduling algorithm, priority is associated with each process and on the basis of that priority each process is executed. There is a problem of starvation in this algorithm. Now see the performance of the proposed algorithms. The performance of SJRR is not very good as compared to the optimal SJF scheduling algorithm but when it is compared with other algorithms it gives good performance that can be seen in Chapter 4. It gives minimum average waiting time and turnaround time as compared to other algorithms. There is no problem of starvation in it and it also removes context switching at much extent. CPU utilization in this algorithm is also high. As working of this algorithm is same as Round Robin so from comparison of proposed algorithm with Round Robin it can be clearly seen that SJRR is a good enhancement. When performance of Combinatory algorithm is compared with other existing algorithms and SJRR it has been seen that it gives good results than FCFS, RR, Priority and SJRR but its performance is slightly less than SJF. It gives minimum average waiting time, turnaround time and maximum CPU utilization. In this algorithm, problem of starvation has been removed at much extent. It has been cleared from the results and whole discussion that proposed algorithms have got its position above than the middle algorithms.

In this work a simulator called CPU Scheduling Algorithm Simulator for proposed scheduling algorithms as well as existing scheduling algorithms has also been developed. Deterministic evaluation of CPU scheduling algorithms can be performed by using this

simulator. Along with performance of proposed and existing scheduling algorithms can be measured with this simulator. This simulator can be used in the class room for students to understand CPU scheduling algorithms in detail. Following are the CPU scheduling algorithms that are simulated using this simulator:

- First Come First Served (FCFS) Scheduling Algorithm
- Shortest Job First (SJF) Scheduling Algorithm
- Round Robin (RR) Scheduling Algorithm
- Priority Scheduling Algorithm
- SJRR Scheduling Algorithm
- Combinatory Scheduling Algorithm

A user friendly Graphical User Interface has been developed that provides an opportunity to the user to save all the input of the process in the database and then select the algorithm from given choice for execution. The Gantt chart of all the processes has been displayed in GUI. A message box is also shown against each algorithm to display average waiting time and average turnaround time.

5.2 Future Work

This scheduling system has been designed to analyze the performance of proposed scheduling algorithms and existing scheduling algorithms. As in this system resources of single physical machine have been used; software module can be developed to analyze CPU scheduling in multiprocessing system.

In this system only non-preemptive scheduling algorithms are implemented, preemptive scheduling algorithms can be included in this system. The system is static (no process can enter into the system at run time); it can be made dynamic with some improvements.

References

1. Milan Milenkovic, “Operating Systems Concepts and Design”, Second Edition, McGraw Hill, IBM Corporation
2. Silberschatz A., Galvin P., and Gagne G., “Operating System Concepts”, Sixth Edition
3. Sonal Sood, Pramod Barthwal, “Simulation of Process Scheduling Algorithms”, Florida International University, Miami’s Public Research University
4. Andrew S.Tanenbaum, Albert S. WoodHull, “Operating Systems Design and Implementation”Second Edition
5. Akhtar Hussain, “Optimized Performance of CPU Scheduling”, College of Electrical and Mechanical Engineering, National University of Science and Technology, Islamabad
6. H.M. Deitel, “Operating Systems” , Second Edition, Addison- Wesley Publishing Company
7. http://en.wikipedia.org/wiki/Scheduling (computing).html
8. Michael Pinedo, “Scheduling:theory, algorithms and systems”
9. http://www.eee.metu.edu.tr/~halici/courses/442/Ch2%20Process%20Scheduling.pdf
10. http://people.msoe.edu/~taylor/cs384/mellottk.pdf
11. www.cs.rutgers.edu/~iftode/cs416_08_10.ppt
12. Beck L.L, Addison Wesley, “System Software”.
13. Gary Nutt, “Operating Systems, A Modern Perspective”, Second Edition
14. http://ece.ut.ac.ir/Classpages/S89/ECE443/slides/5.CPU_Scheduling.pdf
15. www.cs.duke.edu/~chase/cps210-archive/slides/cpu.pdf
16. http://itcs322.almahdi.cc/chap5.doc
17. http://www.scribd.com/doc/7010227/CPU-Scheduling-1
18. http://www.cs.mcgill.ca/~cs310/lect_notes/cs310_lecture06.pdf
19. http://www.cse.buffalo.edu/~bina/cse421/spring02/SchedulingFeb8.pdf
20. Maj. Umar Saleem Butt, “Simulation of CPU Scheduling Algorithms”, College of Electrical and Mechanical Engineering, National University of Science and Technology, Islamabad
21. Saeeda Bibi, Farooque Azam, Sameera Amjad, Wasi Haider Butt, Hina Gull, Rashid Ahmed, Yasir Chaudhry “An Efficient SJRR CPU Scheduling Algorithm” International Journal of

Computer Science and Information Security, Vol. 8, No. 2,2010, pp. 222-230, ISSN: 1947-5500, Pittsburgh, PA 15213, USA

22. Sukanya Suranauwarat, “A CPU Scheduling Algorithm Simulator”, 37th ASEE/IEEE Frontiers in Education Conference
23. E. O. Oyetunji, A. E. Oluleye, “Performance Assessment of Some CPU Scheduling Algorithms”, Research Journal of Information Technology 1(1): 22-26, 2009, ISSN: 2041-3114
24. Rami J. Matarneh, “Self Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes”, American Journal of Applied Sciences 6(10): 18311-1837, 2009, ISSN 1546-9239
25. Mohammed A. F. Husainy, “Best-Job-First CPU Scheduling Algorithm”, Information Technology Journal 6(2): 288-293, 2007, ISSN 1812-5638
26. Syed Nasir Mehmood Shah, Ahmad Kamil Bin Mahmood, Alan Oxley, “Hybrid Scheduling and Dual Queue Scheduling”, 2nd IEEE International Conference on Computer Science and Information Technology, 2009 (ICCSIT 2009), Beijing
27. Bashir Alam, M. N. Doja, R. Biswas, “Finding Time Quantum of Round Robin CPU Scheduling Algorithm Using Fuzzy Logic”, 2008 IEEE Internal Conference on Computer and Electrical Engineering
28. Sami Khuri, Hsiu-Chin Hsu, “Visualizing the CPU Scheduler and Page Replacement Algorithms”, Thirtieth SIGCSE Technical Symposium on Computer Science Education, 1999, New Orleans, Louisiana, United States, ISSN: 0097-8418
29. Sindhu M., Rajkamal R., Vigneshwaran P., "An Optimum Multilevel CPU Scheduling Algorithm", ACE, pp.90-94, 2010 IEEE International Conference on Advances in Computer Engineering, 2010
30. http://en.wikipedia.org/wiki/Rate-monotonic_scheduling