ABSTRACT

The project entitled “CPU Scheduling” is basically a program which simulates the following scheduling algorithms:

First-Come First-served Scheduling, Round-Robin Scheduling, Priority Scheduling, Shortest Job First Scheduling.

CPU Scheduling is a key concept in computer multitasking, multiprocessing operating system and real time operating system designs. It refers to the way processes are assigned to run on the available CPUs, since there are typically many more processes running than there are available **CPUs.**
CPU Scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU. By switching the CPU among processes, the operating system can make the computer more productive. A multiprogramming operating system allows more than one process to be loaded into the executable memory at a time and for the loaded process to share the CPU using time-multiplexing.
Scheduling algorithm is the method by which threads, processes or data flows are given access to system resources (e.g. processor time, communications bandwidth). The need for a scheduling algorithm arises from the requirement for most modern systems to perform multitasking (execute more than one process at a time) and multiplexing (transmit multiple flows simultaneously).

1. INTRODUCTION

1.1 CPU SCHEDULING

1.1.1 Scheduling

An operating system is a program that manages the hardware and software resources of a computer. It is the first thing that is loaded into memory when we turn on the computer. Without the operating system, each programmer would have to create a way in which a program will display text and graphics on the monitor. The programmer would have to create a way to send data to a printer, tell it how to read a disk file, and how to deal with other programs.

In the beginning, programmers needed a way to handle complex input/output operations. The evolution of computer programs and their complexities required new necessities. Because machines began to become more powerful, the time a program needed to run decreased. However, the time needed for handling off the equipment between different programs became evident and this led to programs like DOS. This confirms that operating systems were originally made to handle these complex input/output operations like communicating among a variety of disk drives.

In a multiprogramming system, multiple processes exist concurrently in main memory. Each process alternates between using a processor and waiting for some event to occur, such as the completion of an I/O operation. The processor or processors are kept busy by executing one process while the others wait, hence the key to multiprogramming is **scheduling.**

1.2 Goal & Objective of the Project

The implementation of the CPU Scheduling algorithms is an automation that provides efficient and errorless computation of waiting times, turnaround times, finishing times and normalized turnaround times of First Come First Served (FCFS), Shortest Job First (SJF), Round Robin (RR) and Priority Scheduling algorithms. The system provides a clean and convenient way to test the given data and do the analysis of the CPU scheduling algorithms mentioned above.

1.3 Scheduling Concepts:

- **Long-term Scheduling:**

In a batch system, there are often more processes submitted than can be executed immediately. These processes are spooled to a mass-storage device (typically a disk), where they are kept for later execution. The **long-term scheduler** (or the **job scheduler**) selects processes from this pool and loads them into memory for execution.

The long-term scheduler executes much less frequently. The long-term scheduler controls the **degree of multiprogramming**. Thus, the long-term scheduler may need to be invoked only when a process leaves the system.

- **Medium-term Scheduling:**

The key idea behind a **medium-term scheduler** is that sometimes it can be advantageous to remove processes from memory (and from the active contention for the CPU), and thus to reduce the degree of multiprogramming. At some later time, the process can be reintroduced into memory and its execution can be continued where it left off. This scheme is called **swapping.** The process is swapped out and swapped in later by the medium-term scheduler.

- **Short-term Scheduling:**

The objective of multiprogramming is to have some process running at all time, to maximize CPU utilization. A process is executed until it must wait, typically for the completion of some I/O request. When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process. This pattern continues. Every time one process has to wait, another process may take over the use of the CPU.

Scheduling is a fundamental operating-system function. Almost all computer resources are scheduled before use. The CPU is, of course, one of the primary computer resources. Thus, its scheduling is central to operating-system design.

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

Preemptive / Non-preemptive Policies:

In a **preemptive** scheme, the short-term scheduler may remove a process from the running state in order to allow another process to run. Under **non-preemptive** scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

Non-preemptive scheduling is the only method that can be used on certain hardware platforms, because it does not require the special hardware (for example, a timer) needed for preemptive scheduling.

Scheduling Criteria:

In making decisions about the scheduling of processor work, a number of criteria can be taken into account by the operating system. Many criteria have been suggested for comparing the CPU scheduling algorithms. The criteria to be considered include the following:

- **CPU utilization:** CPU must be kept as busy as possible.
- **Throughput:** It is the number of processes that are completed per unit time.
- **Turnaround time:** It is interval from the time of submission of a process to the time of completion. It is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.
- **Waiting time:** Waiting time is the sum of the periods spent waiting in the ready queue.
- **Response time:** It is the time from the submission of a request until the first response is produced. This is the amount of time it takes to start responding, but not the time that it takes to output that response.

2. Literature Survey

2.1 Existing System

Existing system of CPU Scheduling in XP

- Priority-based, preemptive scheduling
- Scheduling on the basis of threads
- Highest priority thread always be dispatched, and runs until:

a) Preempted by a higher priority thread

b) It terminates

c) Its time quantum ends

d) It makes a blocking system call (e.g., I/O)

32 priorities, each with own queue:

Memory mgmt.: 0; variable: 1 to 15; real-time: 16 to 31

Existing system of CPU Scheduling in LINUX

- Priority-based, preemptive
- Two priority ranges (real time and nice)
- Time quantum longer for higher priority processes (ranges from 10ms to 200ms)
- Tasks are run able while they have time remaining in their time quantum; once exhausted, must wait until others have exhausted their time quantum

Rialto was designed and built to combine the benefits of today's desktop operating systems with the predictability of the best soft real-time systems. While a number of systems have explored mixing timesharing and real-time workloads a number of problems with them led us to design and build a new scheduler.

Some systems supporting mixed timesharing and real-time workloads provide timely execution of real-time applications through proportional share CPU allocation mechanisms based on weighted fair queuing or through fixed share CPU allocation. But these systems have provided no means to guarantee in advance that specific tasks will meet their deadlines when the tasks may require more CPU time than their callers' CPU shares can ensure, or when they will not start until a time in the future.

Other systems have implemented deadline-based time constraints or similar deadline scheduling mechanisms. However, these constraints cannot be guaranteed in advance in the absence of or with insufficient ongoing CPU resource reservations or shares. Furthermore, in, depending upon priorities and admission control, new time constraints can steal time that might otherwise have been needed to finish an existing constraint on time or to maintain other applications' proportional share requirements. Constraints could steal time in our earlier work as well. In our present system, we wanted to provide stronger application independence guarantees.

Finally, many commercial systems have provided fixed-priority scheduling for real-time tasks in addition to round-robin scheduling for timesharing, often with the drawback of the possibility of starving timesharing tasks, while still providing no guarantees for real-time tasks unless they are executed at the highest priority.

2.2 Proposed System:

- When designing an operating system, a programmer must consider which scheduling algorithm will perform best for the use the system is going to see.
- There is no universal "best" scheduling algorithm, and many operating systems use extended or combinations of the scheduling algorithms above.
- For example, Windows NT/XP/Vista uses a Multilevel feedback queue, a combination of fixed priority preemptive scheduling, round-robin, and first in first out. In this system, processes can dynamically increase or decrease in priority depending on if it has been serviced already, or if it has been waiting extensively.
- Every priority level is represented by its own queue, with round-robin scheduling amongst the high priority processes and FIFO among the lower ones. In this sense, response time is short for most processes, and short but critical system processes get completed very quickly. Since processes can only use one time unit of the round robin in the highest priority queue, starvation can be a problem for longer high priority processes.
- In this project, the Scheduling algorithms are designed using C graphics

3. System Specification

3.1 Hardware and Software Requirements

Hardware Requirements

Processor	:	Intel Pentium IV or more
Ram	:	512 MB or more
Cache	:	512 KB
Hard disk	:	16 GB hard disk recommended for primary partition.
Graphics display card	:	VGA or SVGA

Software Requirements

Operating system	:	Windows 7 or later (IIS/SMTP included)
Front End Software	:	Turbo C
User interface	:	CGI

3.2 Programming in C:

Every full C program begins inside a function called "main". A function is simply a collection of commands that do "something". The main function is always called when the program first executes. From main, we can call other functions, whether they be written by us or by others or use built-in language features. To access the standard functions that comes with your compiler, you need to include a header with the #include directive. What this does is effectively take everything in the header and paste it into your program. Let's look at a working program:

```
#include <stdio.h>
int main()
{
printf( "I am alive!  Beware.\n" );
getchar();
return 0;
}
```

Let's look at the elements of the program. The #include is a "preprocessor" directive that tells the compiler to put code from the header called stdio.h into our program before actually creating the executable. By including header files, you can gain access to many different functions--both the printf and getchar functions are included in stdio.h.

The next important line is intmain(). This line tells the compiler that there is a function named main, and that the function returns an integer, hence int. The "curly braces" ({ and }) signal the beginning and end of functions and other code blocks. If you have programmed in Pascal, you will know them as BEGIN and END. Even if you haven't programmed in Pascal, this is a good way to think about their meaning.

The printf function is the standard C way of displaying output on the screen. The quotes tell the compiler that you want to output the literal string as-is (almost). The '\n' sequence is actually treated as a single character that stands for a newline (we'll talk about this later in more detail); for the time being, just remember that there are a few sequences that, when they appear in a string literal, are actually not displayed literally by printf and that '\n' is one of them. The actual effect of '\n' is to move the cursor on your screen to the next line. Notice the semicolon: it tells the compiler that you're at the end of a command, such as a function call. You will see that the semicolon is used to end many lines in C.

The next command is getchar(). This is another function call: it reads in a single character and waits for the user to hit enter before reading the character. This line is included because many compiler environments will open a new console window, run the program, and then close the window before you can see the output. This command keeps that window from closing because the program is not done yet because it waits for you to hit enter. Including that line gives you time to see the program run.

Finally, at the end of the program, we return a value from main to the operating system by using the return statement. This return value is important as it can be used to tell the operating system whether our program succeeded or not. A return value of 0 means success. The final brace closes off the function. You should try compiling this program and running it. You can cut and paste the code into a file, save it as a .c file, and then compile it. If you are using a command-line compiler, such as Borland C++ 5.5, you should read the compiler instructions for information on how to compile. Otherwise compiling and running should be as simple as clicking a button with your mouse (perhaps the "build" or "run" button).

3.3 Features of C:

C is a MIDDLE LEVEL LANGUAGE because it combines the elements of HIGH LEVEL LANGUAGE with the functionalism of ASSEMBLY LANGUAGE .C allows manipulation of bits, bytes and addresses, which are the basic elements with which the computer functions. Also, C code is very portable, that is software written on one type of computer can be adapted to work on another type. Although C has 5 basic built-in data types, it is not strongly typed language compared to high-level languages. C permits almost all data type conversions.

C is an ideal for system-level programming. The term block structure language does not apply strictly to C. Technically a block structure language permits procedure and functions to be declared inside other procedures of functions.

C does not allow the creation of functions within functions, and therefore cannot be formally called a block-structured language.

C allows compartmentalization of code and data. This is a distinguishing feature of any structured language. It refers to the ability of a language to section of and hides all information and instructions necessary to perform a specific task from the rest of the program. Code can be compartmentalized in C using functions or code blocks. FUNCTIONS are used to define and code separately, special tasks required in the program. This allows program to be modular. CODE BLOCK is a logically connected statement that is treated like a unit. Placing a sequence of between opening and closing flower braces creates a code block. Also a structured language supports several loop constructs, such as while, do-while, and for. It allows the user to indent statements and does not require a field concept. C was initially used for system programming. A system program forms a portion of the operating system of the computer of its support utilities. Operating system, Interpreters, Editors, Assembly programs are usually called system program.

The UNIX Operating System was developed using C. Today C used by many programmers to program all kind of task because of its portability and efficiency. There is C compilers available for almost all computers. A code written in C on a particular machine can compiled and run on another machine by making a few or no changes. Also C compiler produces very fast object code. In addition to the above, C offers the speed of an assembly language, but few of the restrictions of a structured language. Programmers can create and maintain a unique library of functions, which can be used by many different programs. Thus large projects can be managed easily, with minimal duplication of effort.

3.4 Introduction of Graphics:

In today's world of advance technology, interactive computer graphics has become a powerful tool for production of realistic features. Today, we find computer graphics used in various areas that include science, engineering, medicine, industry, art, entertainment etc. The main reason for the effectiveness of computer graphics is the speed with which the user can understand the displayed information. The graphics in Turbo C provides a wide variety of built in functions. Computer Graphics remains one of the most exciting and rapidly growing computer fields. It has become a common element in the user interface, data visualization, TV commercials, motion pictures and many other applications.

The current trend of computer graphics is to incorporate more physics principles into 3-D graphics algorithm to better simulate the complex interactions between objects and lighting environment.

Turbo C includes an extensive collection of graphics-oriented library functions. These functions fall into two categories:

1. Those that work in text mode.
2. Those that work in graphics mode

Text Mode Function:

- The text mode functions are concerned with placing text in certain areas of screen.
- They work in any graphics monitor and adapter.
- In this mode, the screen is divided into character positions, typically 80 columns and 25 rows.
- It also facilitates writing text to windows.

Graphics Mode Function:

- Graphics mode requires a graphics monitor and adapter card such as CGA, EGA and VGA.
- It allows one to draw dots, lines, and shapes, add colors to lines and areas and perform many other graphics related activities.
- In graphics mode, one can address individual pixels, which gives a much finer resolution.
- When we start drawing any graphics on the screen we need the header file called GRAPHICS.H and library file called GRAPHICS.LIB
- The header file contains definitions and explanations of all the functions and explanation of all the functions and constants needed in the program, and the graphic function are kept in the graphics library file. Both these functions are kept in the graphics library file. Both these files provide as a part of Turbo C++ package.
- To switch to over graphics mode that offers the best resolution, the initgraph() function is called. It figures out the best resolution and puts the corresponding number to that mode in the variable gm. The gm

number tells the monitor the resolution we are using, the number of video pages it supports and the colors that are available.

- A variable gd can be assigned to DETECT, which asks the initgraph() to figure out the BGI file needed and load it into the memory.
- When switched to the graphics mode, firstly the cursor disappears since the graphics mode doesn't support the conventional cursor.
- Secondly, the co-ordinate system is established whereby the top-left corner of the screen is treated as origin (0, 0). The X-axis goes horizontally across, and the Y-axis goes vertically downwards.
- In the graphics mode users can use functions like line(), circle(), rectangle() etc for drawing and functions like setfillstyle(), setcolor(), floodfill() can be used to fill the colors.
- Turbo C also provides generation of sounds using three functions namely sound(), delay() and nosound(). These files are defined in the DOS.H header file.
- When exiting from graphics mode the system should be restored to the previous video mode. To restore the previous mode abd release the memory used by the graphics mode, the closegraph() library function should be executed.

By 1960, a hoard of computer languages had come into existence, almost each for specific purposes. COBOL was being used for commercial applications, FORTRAN for Engineering and Scientific applications and so on. Instead of learning and using so many languages, each for a different purpose, an international committee was set up to develop a language which can program all possible applications. This committee came out with a language called ALGOL 60, which was too abstract and general. BCPL was developed by Martin Richards aimed to solve the problems of the previous languages. Around same time 'B' was developed by Ken Thomson at AT&T's Bell Labs. Finally 'C' was originally developed in 1970's by Dennis Ritchie at Bell Laboratories, Inc. It is an outgrowth of BCPL and B.

Graphics Primitives

A primitive is a graphics object that is essential for the creation or construction of complex images. Fortunately, graphics is constructed from three basic elements, as opposed to the great variety of graphics applications. The most basic of these elemental structures is the pixel, short for picture element.

Pixel:

A pixel is a point of light. It is just one tiny dot on the raster displays. Though it has no structure, it is an essential building block and hence it can be considered as the graphics primitive.

The resolution of CRT is related to the dot size, the diameter of a single dot. A resolution of 100 dots lines/inch implies a dot size of 0.01 inch. However, in reality, pixels are more elliptic than circle. The shape of a pixel purely depends upon the characteristics of the visual display unit. The ratio of the distance between the centers of two adjacent horizontal pixels to that of the vertical ones is called the pixel ratio.

For example, if 100-pixel lines in x and y measure 6cm and 8cm respectively, then

Pixel ratio should be considered in line-generating algorithms.

Line:

Line, especially straight lines, constitutes an important building block of computer images. For example, Line is the basic building block of Line graphs, bar and pie charts, two and three-dimensional graphs of mathematical functions, engineering drawings and architectural plans.

In computer graphics, straight line is so basic in creating images that we call it a graphics primitive. Straight lines can be developed in two different ways. A structural method determines which pixels should be set before drawing the line; a conditional method tests certain conditions to find which pixel should be set next.

Image Types:

Two-dimensional:

2D computer graphics are the computer-based generation of digital images—mostly from two-dimensional models, such as 2D geometric models, text, and digital images, and by techniques specific to them.

2D computer graphics are mainly used in applications that were originally developed upon traditional printing and drawing technologies, such as typography, cartography, technical drawing, advertising, etc.. In those applications, the two-dimensional image is not just a representation of a real-world object, but an independent artifact with added semantic value; two-dimensional models are therefore preferred, because they give more direct control of the image than 3D computer graphics, whose approach is more akin to photography than to typography.

Three-dimensional:

3D computer graphics in contrast to 2D computer graphics are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering 2D images. Such images may be for later display or for real-time viewing.

Despite these differences, 3D computer graphics rely on many of the same algorithms as 2D computer vector graphics in the wire frame model and 2D computer raster graphics in the final rendered display. In computer graphics software, the distinction between 2D and 3D is occasionally blurred; 2D applications may use 3D techniques to achieve effects such as lighting, and primarily 3D may use 2D rendering techniques.

3D computer graphics are often referred to as 3D models. Apart from the rendered graphic, the model is contained within the graphical data file. However, there are differences. A 3D model is the mathematical representation of any three dimensional object. A model is not technically a graphic until it is visually displayed. Due to 3D printing, 3D models are not confined to virtual space. A model can be displayed visually as a two-dimensional image through a process called 3D rendering, or used in non-graphical computer simulations and calculations. There is some 3D computer graphics software for users to create 3D images.

4. SYSTEM DESIGN

4.1 Operating system

An operating system is an essential component of a computer system. The primary objective of an operating system is to make the computer system easy to use and utilize computer hardware in an efficient manner. The operating system controls and coordinates the use of hardware among the various application programs for the various users.

An operating system is an argue collection of software, which manages resources of the computer such as memory, processor, and file system on input output devices. Like a government it keeps tracks of the status of each resource and decides who will have a control over the computer resources, for how long and when.

4.1.1 The main objective of operating system:

- Increase the productivity of processing resources.
- Make computer system convenient to use
- Use computer hardware in an efficient manner

4.1.2 Ways to interact with the O.S:

- By means of operating system call in a program.
- Directly by means of operating system commands.

4.1.3 Types of operating system

- Batch operating system.
- Multiprogramming operating system.
- Network operating system.
- Distributed operating system.

4.2 SYSTEM ANALYSIS OF OS

System analysis is the most powerful phase of development. In analysis phase, one has to study the existing system in detail and has to collect necessary information, regarding the system to be designed. After analyzing the data and studying the existing, the new system is developed, which in turn improves developing practices.

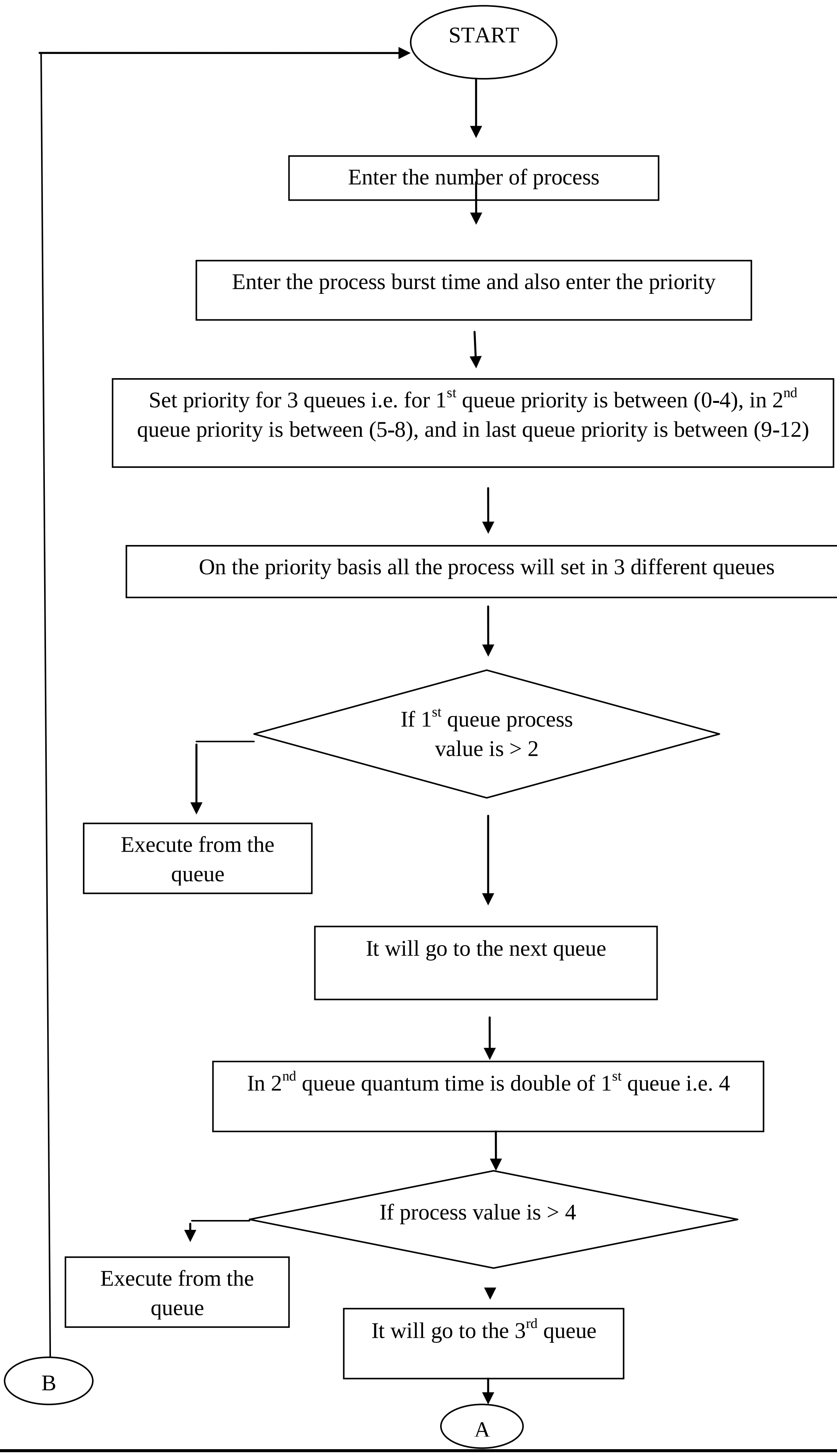
4.2.1 Salient features of the system

The system is a GUI based system. It simulates the process of concept of paging hardware where the user can easily understand the paging concept. The project shows some graphical representation of some paging concept. It shows the different paging techniques with animation. By watching this simulation project a person can easily understand the paging implemented in O.S.

4.2.2 The advantages of the system are as follows

- It's really user friendly
- It's a menu based GUI application
- Dynamic testing of paging hardware
- Static user input to test working principle of paging hardware.

4.3 Project Design


START
Enter the number of process
Enter the process burst time and also enter the priority
Set priority for 3 queues i.e. for 1st queue priority is between (0-4), in 2nd queue priority is between (5-8), and in last queue priority is between (9-12)
On the priority basis all the process will set in 3 different queues
If 1st queue process value is > 2
Execute from the queue
It will go to the next queue
In 2nd queue quantum time is double of 1st queue i.e. 4
If process value is > 4
Execute from the queue
It will go to the 3rd queue
B
A

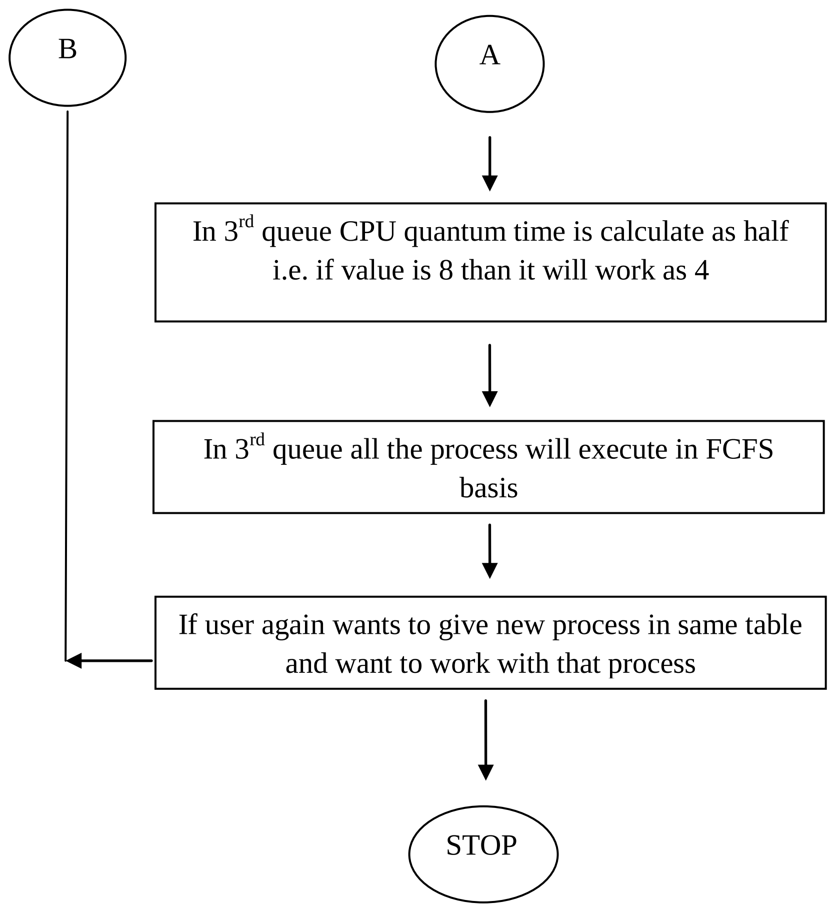

B
A
In 3rd queue CPU quantum time is calculate as half i.e. if value is 8 than it will work as 4
In 3rd queue all the process will execute in FCFS basis
If user again wants to give new process in same table and want to work with that process
STOP

Fig. 4.3.1 priority scheduling

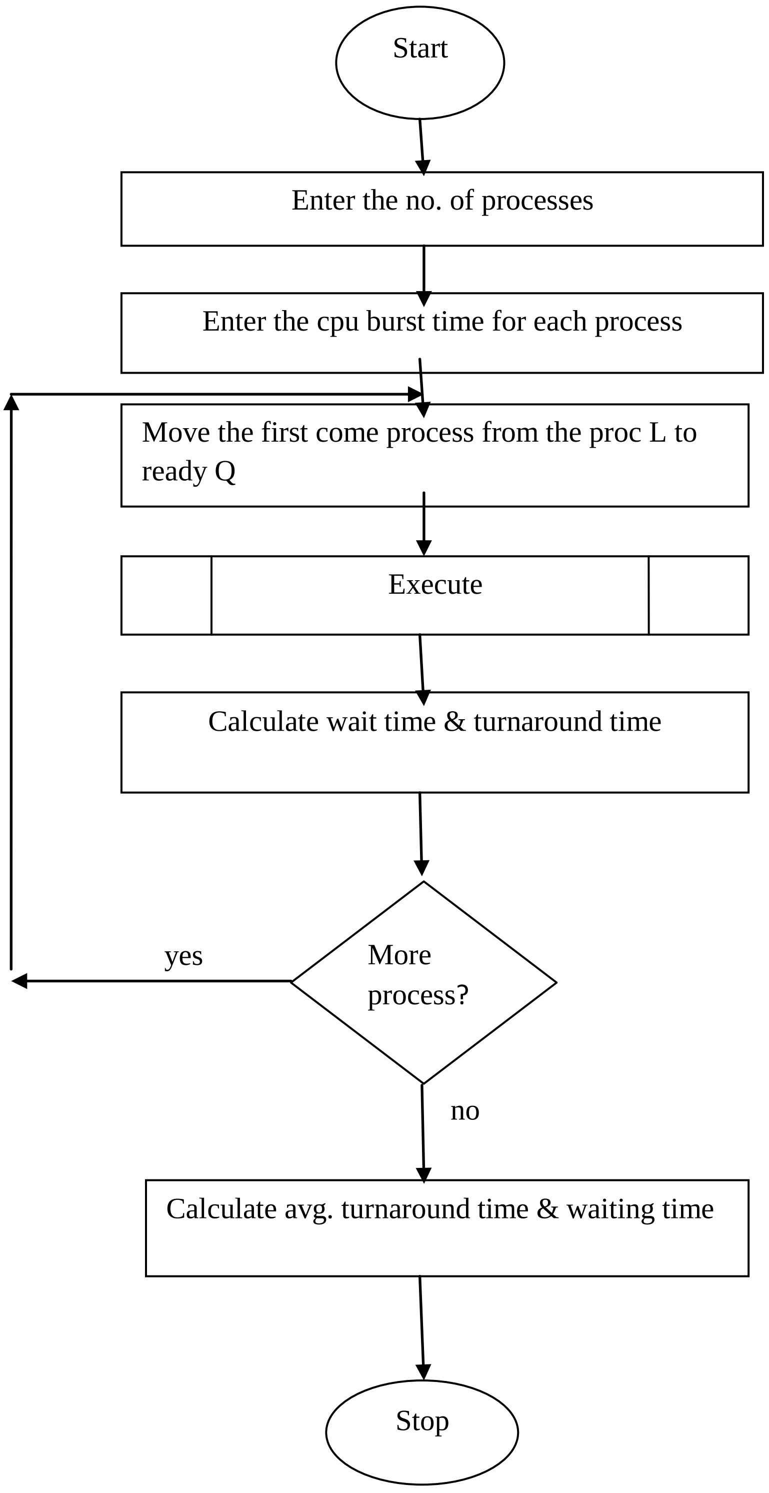

Start
Enter the no. of processes
Enter the cpu burst time for each process
Move the first come process from the proc L to ready Q
Execute
Calculate wait time & turnaround time
More process?
yes
no
Calculate avg. turnaround time & waiting time
Stop

Fig 4.3.2 First come First out

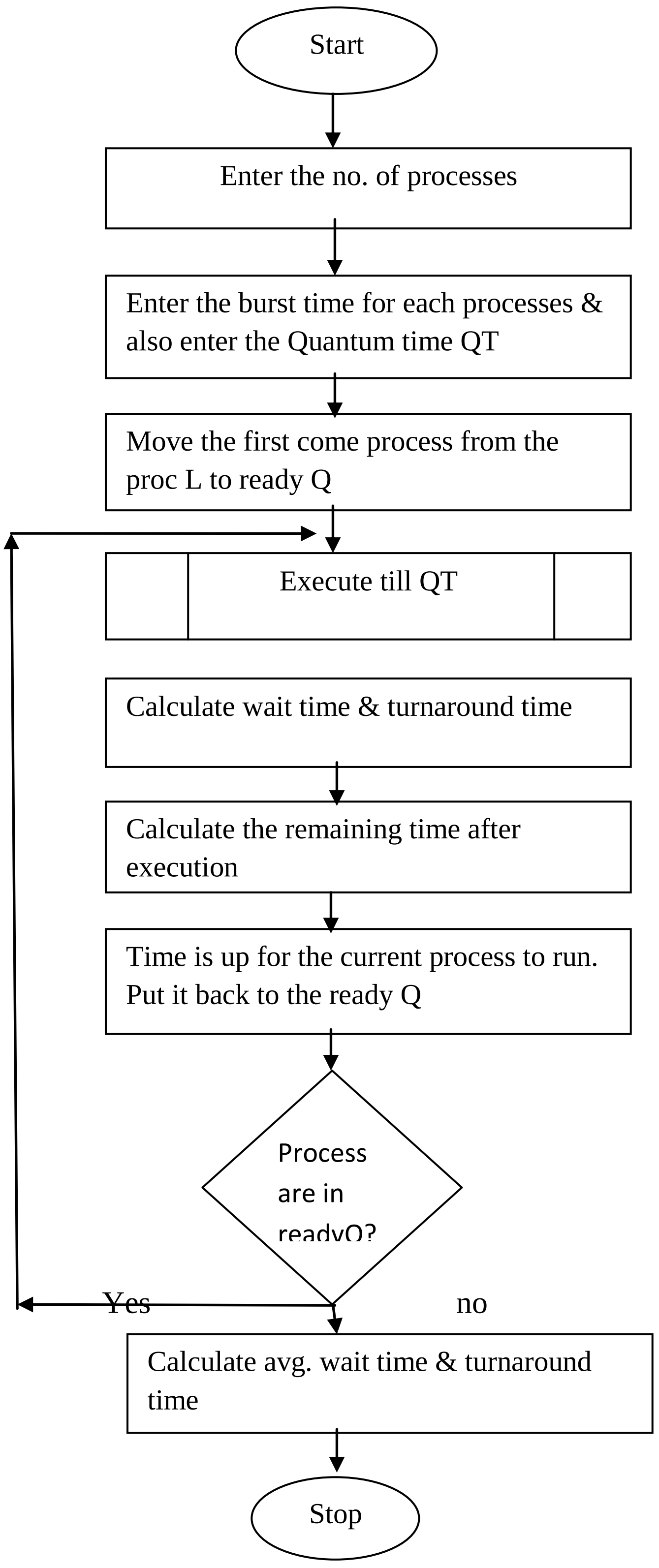

Start
Enter the no. of processes
Enter the burst time for each processes & also enter the Quantum time QT
Move the first come process from the proc L to ready Q
Execute till QT
Calculate wait time & turnaround time
Calculate the remaining time after execution
Time is up for the current process to run. Put it back to the ready Q
Process are in readyQ?
Yes
no
Calculate avg. wait time & turnaround time
Stop

Fig.4.3.3 Round robin

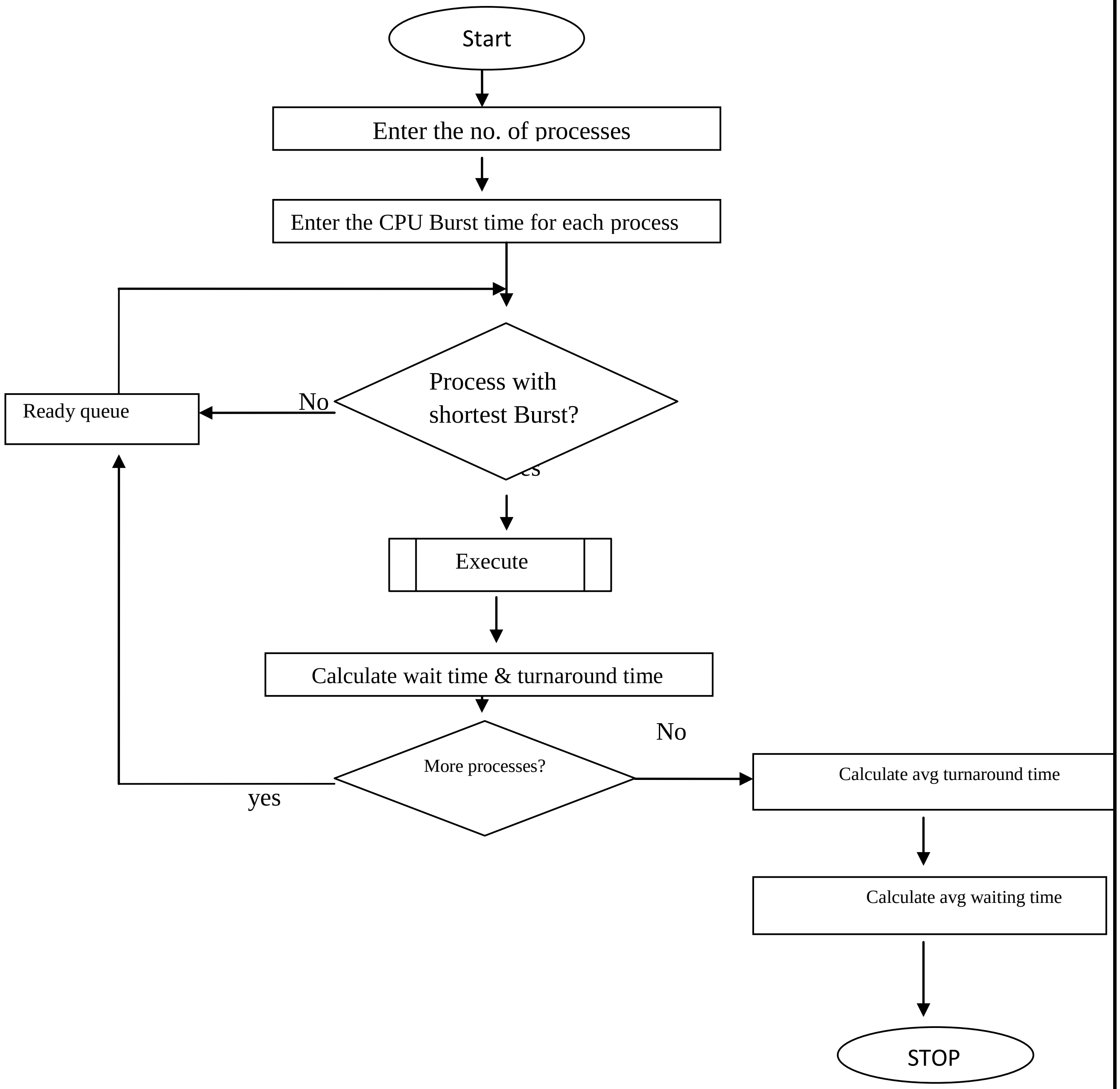

Start
Enter the no. of processes
Enter the CPU Burst time for each process
Process with shortest Burst?
No
Ready queue
Yes
Execute
Calculate wait time & turnaround time
More processes?
yes
No
Calculate avg turnaround time
Calculate avg waiting time
STOP

Fig 4.3.4 shortest job first

5. IMPLEMENTATION

5.1 Scheduling Algorithms:

There are several different scheduling algorithms. Following are the algorithms that have been implemented:

- **First-Come, First-served Scheduling**
- **Round-Robin Scheduling**
- **Priority Scheduling**
- **Shortest Job First Scheduling**

5.1.1 First-Come, First-served Scheduling:

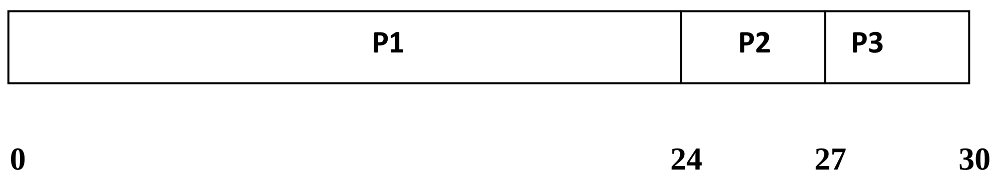
```
Algorithm: FCFS
// Input: Processes with their burst time.
// Output: All processes are executed.
Step 1: New process Pi is initialized.
Step 2: Pi enters ready Queue.
Step 3: Pj is loaded from ready queue to be executed.
Step 4: If CPU is free go to step 5 otherwise go to step 2.
Step 5: Execute the loaded process.
Step 6: Stop.
```

It is by far the simplest CPU scheduling algorithm. With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCFS policy is easily managed with a FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed the queue.

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

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

If the processes arrive in the order P1, P2, P3, and are served in FCFS order, we get the result shown in the following **Gantt chart**:



The waiting time is 0 milliseconds for process P1, 24 milliseconds for process P2, and 27 milliseconds for process P3. Thus, the average waiting time is (0 + 24 + 27) / 3 = 17 milliseconds.

The FCFS scheduling algorithm is non-preemptive. Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU, either by terminating or by requesting I/O.

5.1.2 Round-Robin Scheduling:

The **round-robin (RR)** scheduling algorithm is designed especially for time-sharing systems. It is similar to FCFS scheduling, but preemption is added to switch between processes. A small unit of time, called a **time quantum**, or time slice is defined. The ready queue is treated as a circular queue. The CPU goes scheduler around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.

To implement RR scheduling, we keep the ready queue as a FIFO queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process.

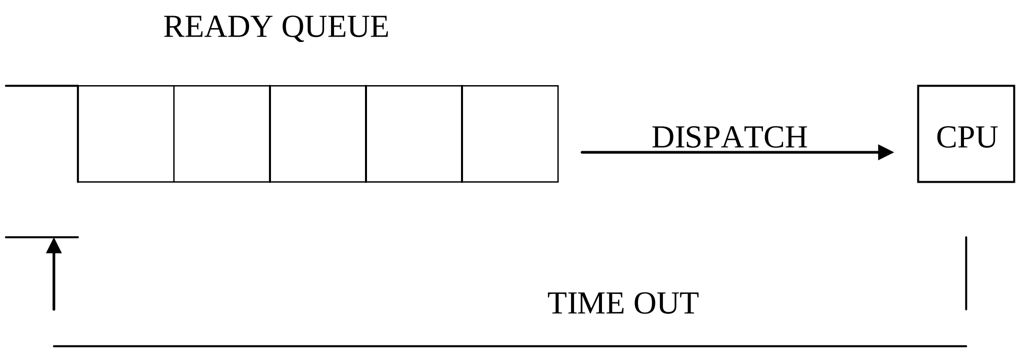
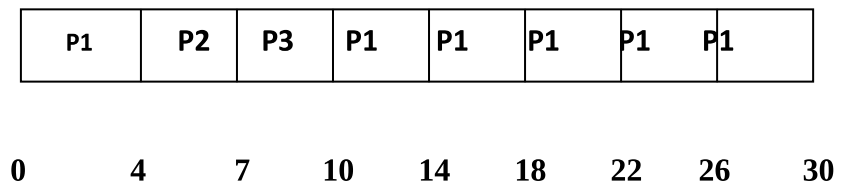


Fig. Round Robin Scheduling

Consider the following set of processes that arrive at time 0, with the length of the CPU burst time given in milliseconds:

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

If we use a time quantum of 4 milliseconds, then process P1 gets the time 4 milliseconds. Since it requires another 20 milliseconds, it is preempted after the first time quantum, and the CPU is given to the next process in the queue, process P2. Since process P2 does not need 4 milliseconds, it quits before its time quantum expires. The CPU is then given to the next process, P3. Once each process has received 1 time quantum, the CPU is returned to process P1 for an additional time quantum. The resultant RR schedule is:



The average waiting time is 17 / 3 = 5.66 milliseconds.

5.1.3 <u>Shortest-Job-First (SJR) Scheduling</u>

```
Algorithm: SJF
// Input: Processes with their burst time.
// Output: All processes are executed.
Step 1: New process Pi is initialized.
Step 2: Pi enters ready Queue.
Step 3: Pj is loaded from ready queue to be executed.
Step 4: If Process is having Shortest Burst time then go to Step 5 otherwise go Step 2.
Step 5: If CPU is free go to step 6 otherwise go to step 2.
Step 6: Execute the loaded process.
Step 7: Stop.
```

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
    - no preemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.
    - Preemptive – if a new process arrives with CPU burst length less than remaining time of current executing

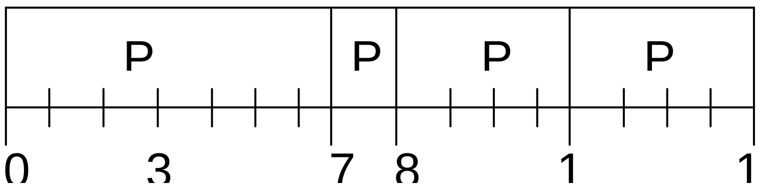
process, preempt. This scheme is known as the Shortest-Remaining-Time-First (SRTF).

- SJF is optimal – gives minimum average waiting time for a given set of processes.

Example of Non-Preemptive SJF

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

SJF (non-preemptive)

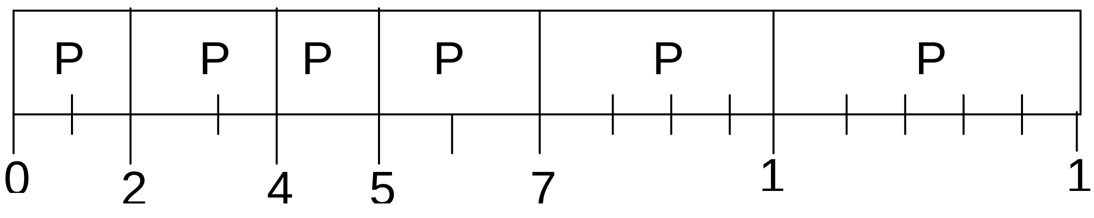


Average waiting time = (0 + 6 + 3 + 7)/4 - 4

Example of Preemptive SJF

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

SJF (preemptive)



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

Determining Length of Next CPU Burst

- Only estimate the length.
- Done by using the length of previous CPU bursts, using exponential averaging.

1. t_n = actual lenght of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define:

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

Examples of Exponential Averaging

α =0

- $\tau_{n+1} = \tau_n$
- Recent history does not count.

$\alpha = 1$

- $\tau_{n+1} = t_n$
- Only the actual last CPU burst counts.

- If we expand the formula, we get:

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

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

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

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

Priority Scheduling

- A priority number (integer) is associated with each process

The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority).

- Preemptive
- No preemptive

- SJF is a priority scheduling where priority is the predicted next CPU burst time.
- Problem ≡ Starvation – low priority processes may never execute.
- Solution ≡ Aging – as time progresses increase the priority of the process.

Round Robin (RR)

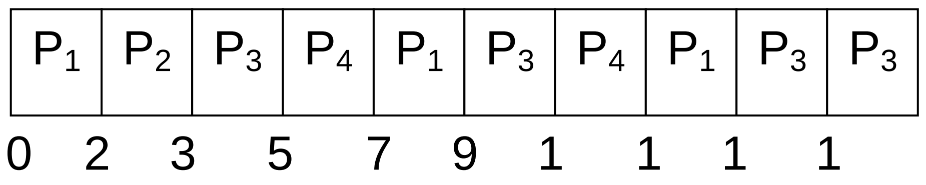
- Each process gets a small unit of CPU time (time quantum), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units at once. No process waits more than (n-1) q time units.
- Performance

- q Large ⇒ FIFO
- q Small ⇒ q must be large with respect to context switch, otherwise overhead is too high.

Example: RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

The Gantt chart is:



Typically, higher average turns around than SJF, but better response.

5.1.4 Priority Scheduling

Algorithm: PS

// Input: Processes with their burst time.

// Output: All processes are executed.

Step 1: New process Pi is initialized.

Step 2: Pi enters ready Queue.

Step 3: Pj is loaded from ready queue to be executed

Step 4: If Process is having Highest then go to Step 5 otherwise go to Step 2.

Step 5: If CPU is free go to step 6 otherwise go to step 2.

Step 6: Execute the loaded process.

Step 7: Stop

Assign a priority p to a process (typically lower p = Higher priority, but this isn't set in stone), and give the CPU to the process with the highest priority.

• Note how SJF is a special case of priority scheduling: p is the inverse of the next predicted CPU burst.

• Processes of equal priority are scheduled FCFS

• Priorities range from internally-calculated metrics (time limits, memory requirements, open files, I/O-to- CPU burst ratio) to external factors

• Comes in both preemptive and cooperative flavors:

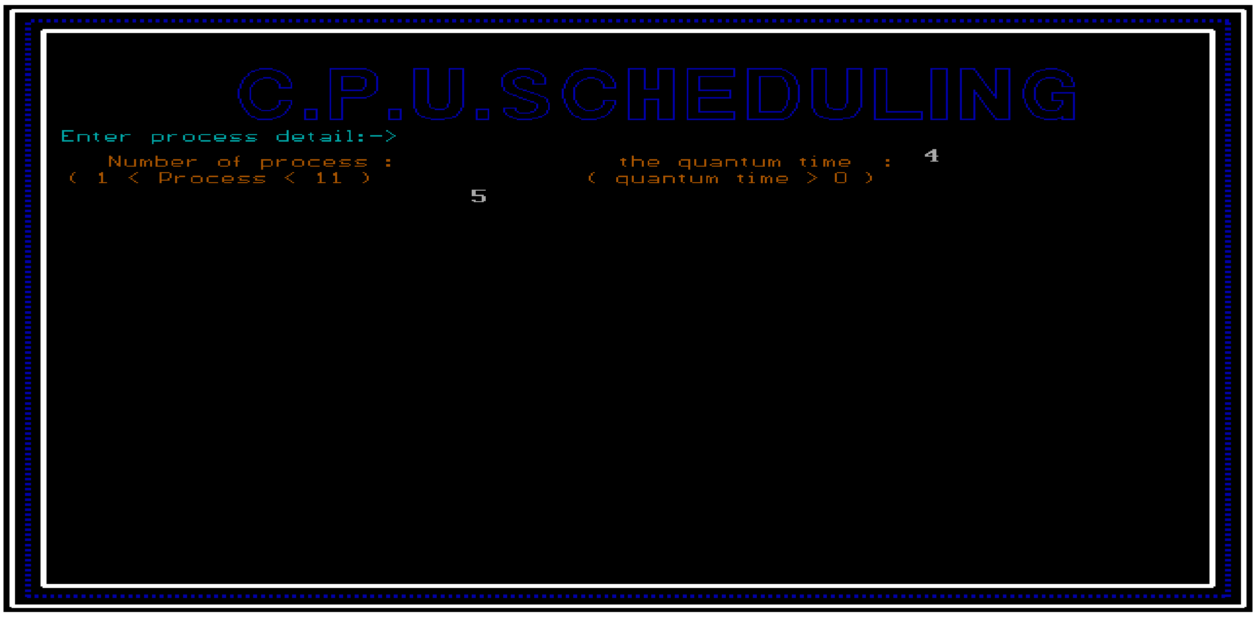
Preemptive version interrupts the currently running process when a higher-priority process comes in Cooperative version puts the higher-priority process at the top of the queue and waits for the currently running process to relinquish the CPU.

• Key issue: indefinite blocking or starvation of a process

low-priority processes may wait forever of higher priority ones keep showing up.

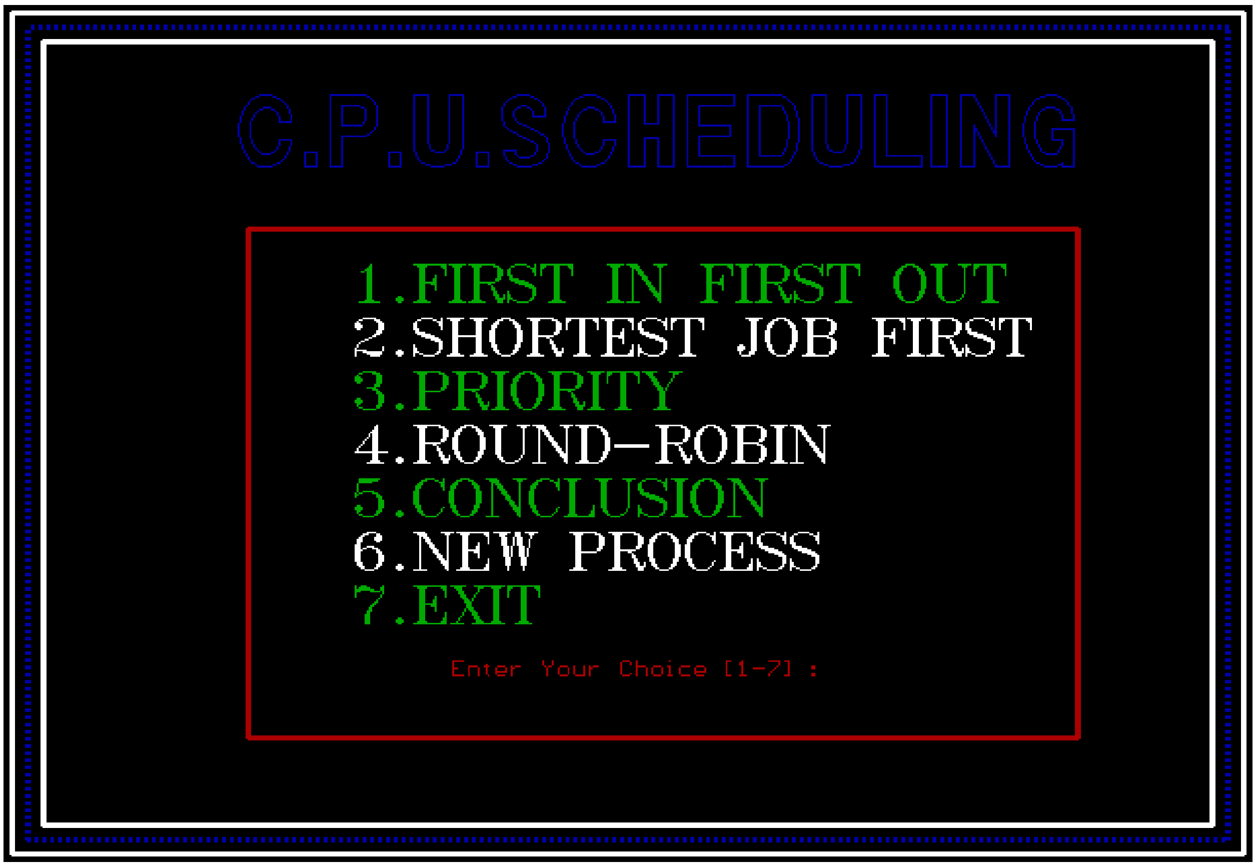
• Address starvation through aging: gradually increase a process's priority as waiting time increases — caps the maximum waiting time.

6. SREENSHORTS



```
C.P.U.SCHEDULING
Enter process detail:->
   Number of process :                    the quantum time  :  4
( 1 < Process < 11 )                    ( quantum time > 0 )
                        5
```

Figure 6.1:-Home Page

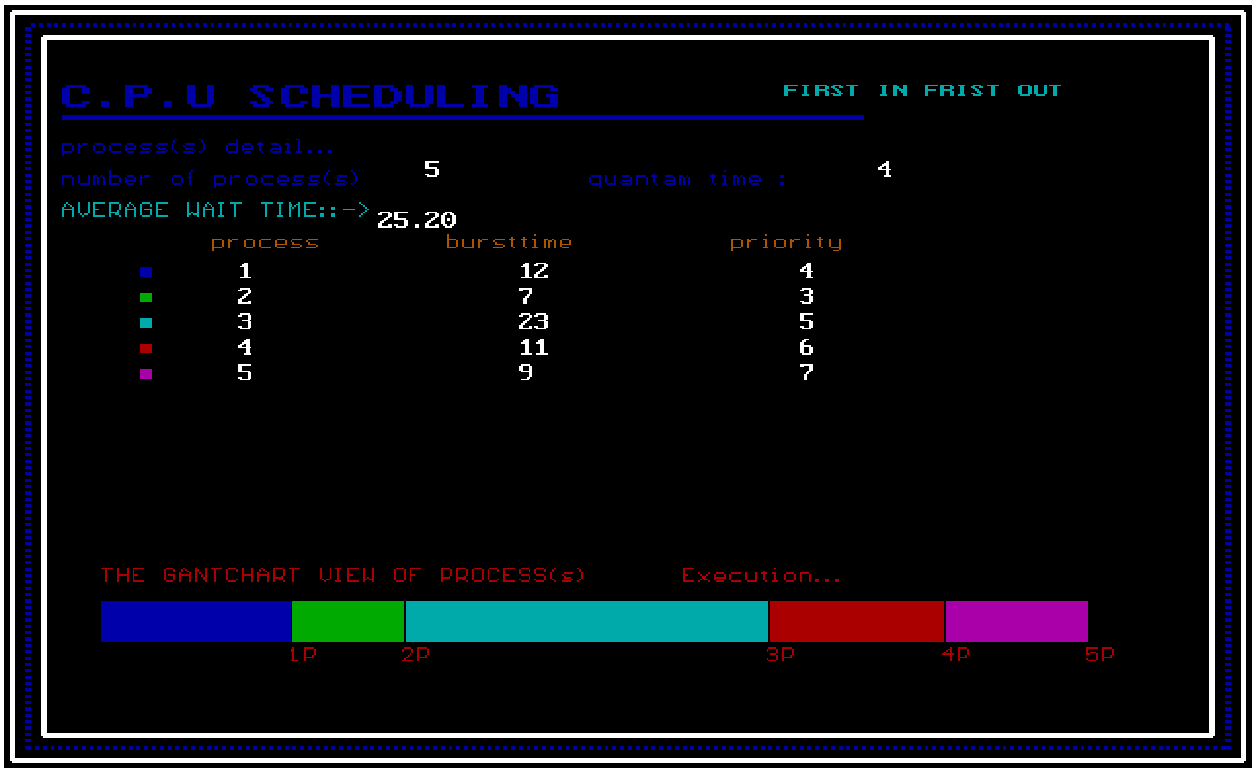


```
C.P.U.SCHEDULING

1.FIRST IN FIRST OUT
2.SHORTEST JOB FIRST
3.PRIORITY
4.ROUND-ROBIN
5.CONCLUSION
6.NEW PROCESS
7.EXIT

Enter Your Choice [1-7] :
```

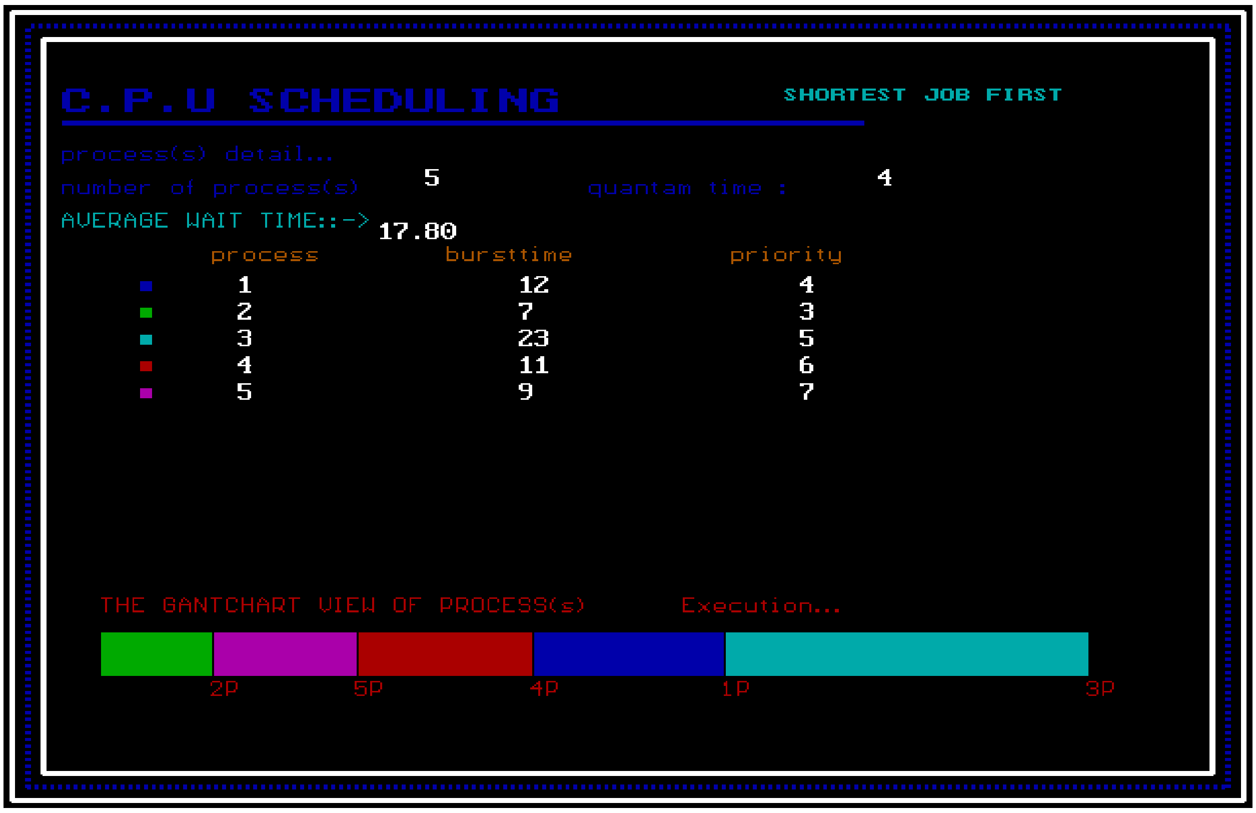
Figure 6.2:-Menu Page



```
C.P.U SCHEDULING                                  FIRST IN FRIST OUT
process(s) detail...
number of process(s)      5            quantam time :          4
AVERAGE WAIT TIME::-> 25.20
              process          bursttime            priority
   🟦         1                   12                   4
   🟩         2                   7                    3
   🟦         3                   23                   5
   🟥         4                   11                   6
   🟪         5                   9                    7

  THE GANTCHART VIEW OF PROCESS(s)         Execution...
                   1P     2P                  3P       4P      5P
```

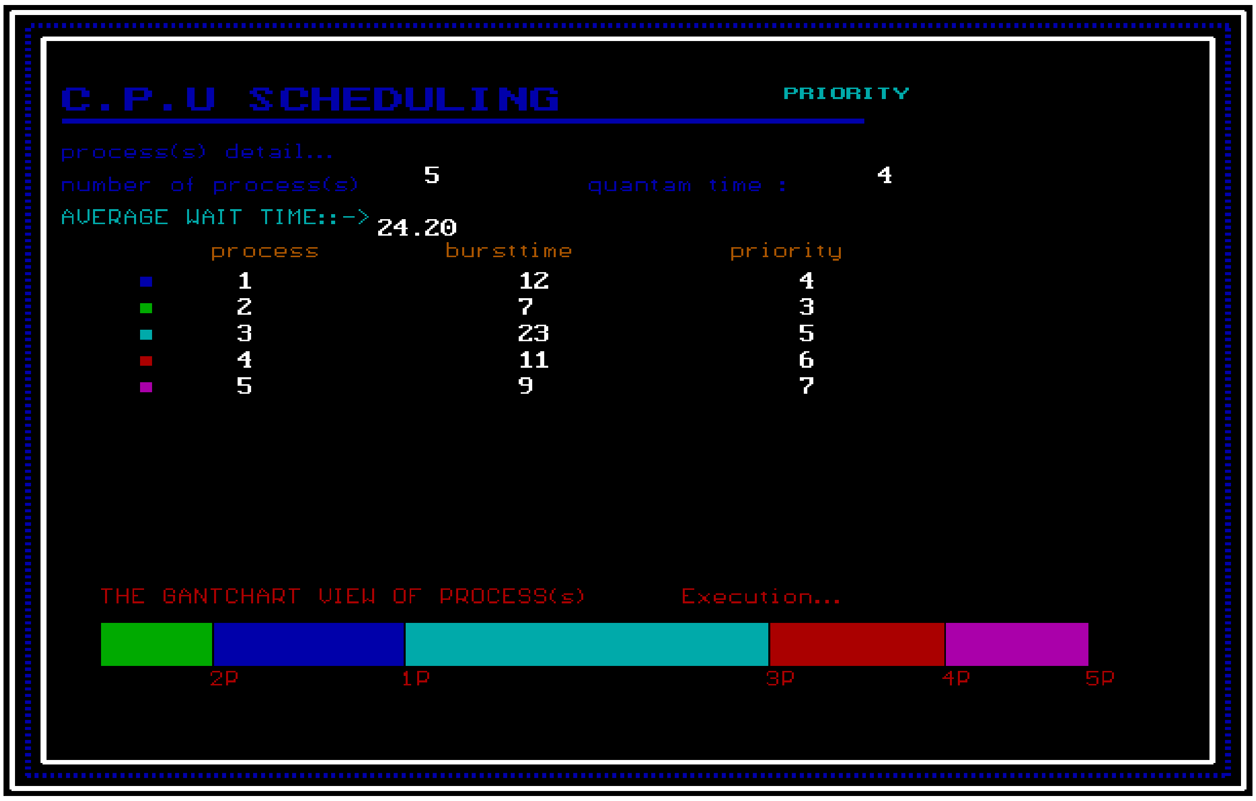
Figure 6.3:-First In First Serve



```
C.P.U SCHEDULING                                  SHORTEST JOB FIRST
process(s) detail...
number of process(s)      5            quantam time :          4
AVERAGE WAIT TIME::-> 17.80
              process          bursttime            priority
   🟦         1                   12                   4
   🟩         2                   7                    3
   🟦         3                   23                   5
   🟥         4                   11                   6
   🟪         5                   9                    7

  THE GANTCHART VIEW OF PROCESS(s)         Execution...
              2P       5P       4P       1P                    3P
```

Figure 6.1:-Shortest Job First

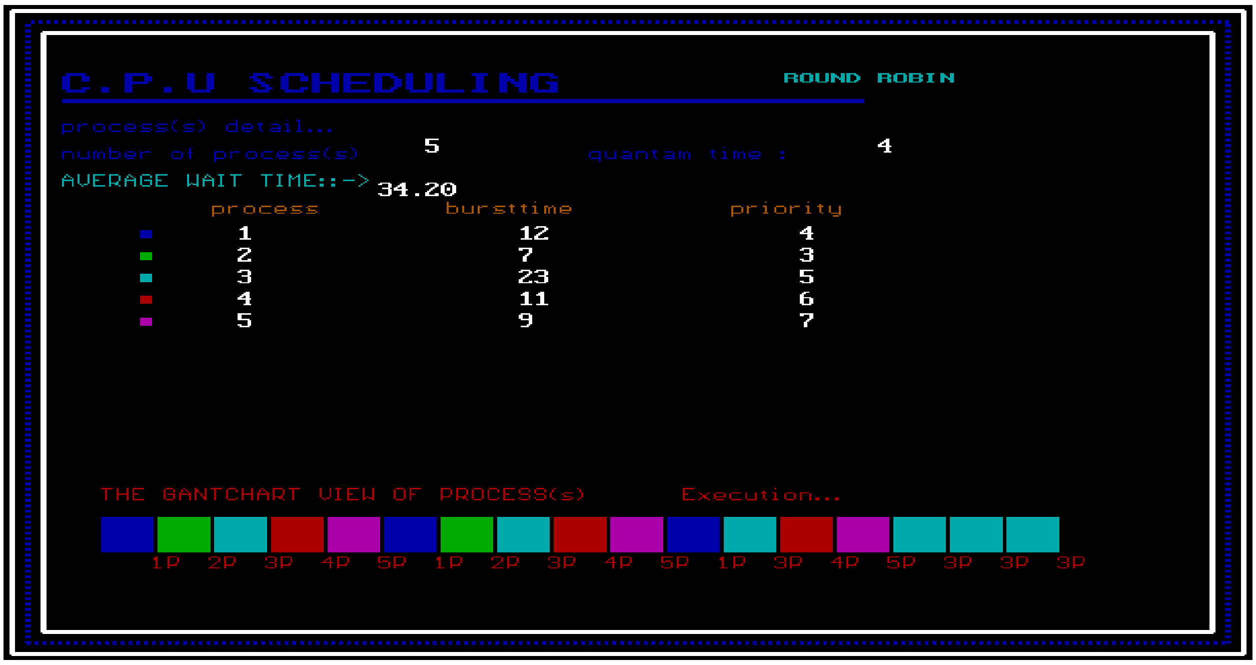


```
C.P.U SCHEDULING                        PRIORITY
process(s) detail...
number of process(s)      5          quantam time :      4
AVERAGE WAIT TIME::-> 24.20
            process          bursttime           priority
   🟦          1                12                  4
   🟩          2                7                   3
   🟦          3                23                  5
   🟥          4                11                  6
   🟪          5                9                   7

   THE GANTCHART VIEW OF PROCESS(s)      Execution...

          2P          1P                   3P         4P       5P
```

Figure 6.1:-Priority Scheduling

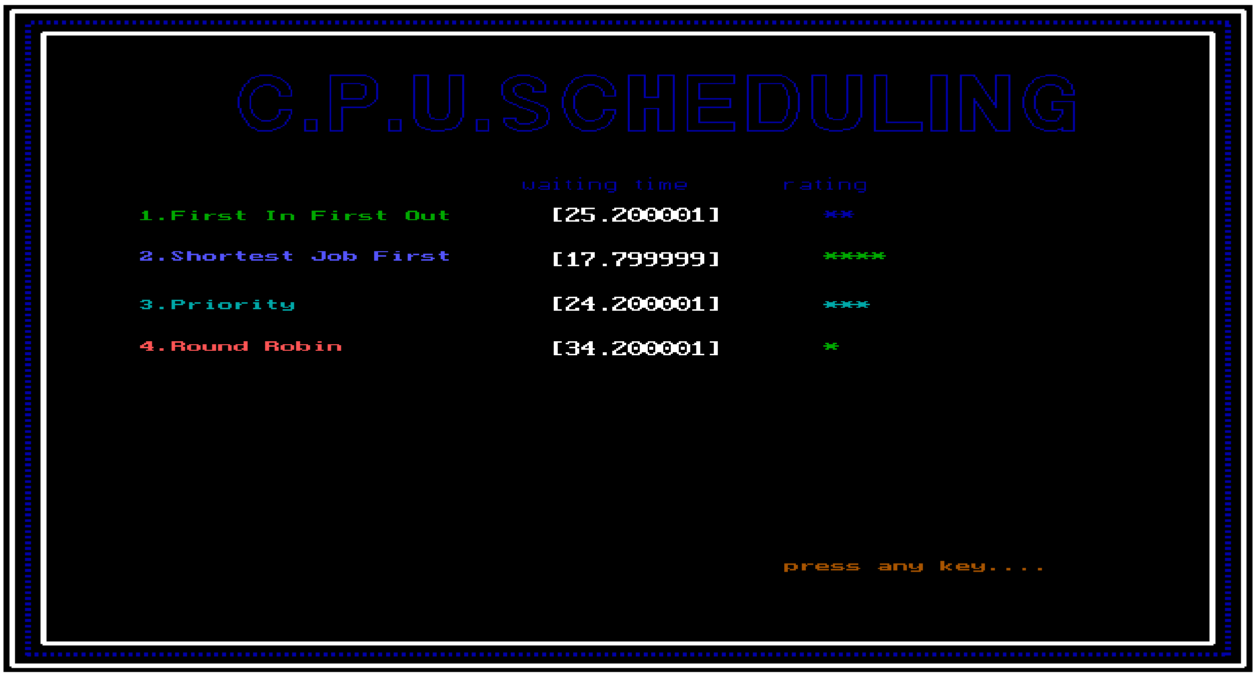


```
C.P.U SCHEDULING                        ROUND ROBIN
process(s) detail...
number of process(s)      5          quantam time :      4
AVERAGE WAIT TIME::-> 34.20
            process          bursttime           priority
   🟦          1                12                  4
   🟩          2                7                   3
   🟦          3                23                  5
   🟥          4                11                  6
   🟪          5                9                   7

   THE GANTCHART VIEW OF PROCESS(s)      Execution...

     1P  2P  3P  4P  5P  1P  2P  3P  4P  5P  1P  3P  4P  5P  3P  3P  3P
```

Figure 6.1:-Round Robin Scheduling



```
                    C.P.U.SCHEDULING

                        waiting time      rating
1.First In First Out     [25.200001]       **
2.Shortest Job First     [17.799999]       *****
3.Priority               [24.200001]       ***
4.Round Robin            [34.200001]       *

                                        press any key....
```

Figure 6.1:-Conclusion

```
C.P.U.SCHEDULING
DONE BY:-
AMRITHA
ANUJA
MUDASIR
****THANK YOU*****
```

Figure 6.1:- Thanking Page

7. APPENDIX

7.1 About C

C is a general purpose programming language. C language is reliable, simple and easy to use. C is also called a systems programming language as it is used for writing compilers and operating systems. It provides various data types such as character, integer, pointers, array, structure and unions.

C provides various control structures, which enable to specify the order in which the various instructions in the program are to be executed by a computer. Control instructions determine the flow of control in a program. There are four types of control structures, they are:

- Sequence control instructions
- Selection or decision control instruction
- Repetition or loop control instruction
- Case control instruction

C language contains various data structures such as arrays, stacks, queues, link, lists and trees. Pointers are a special feature of C.

7.2 Graphics in C

Computer graphics is one of the most powerful and interesting facets of computers. All video games, animations, multimedia predominantly works in graphics. The available graphic modes in Turbo c are:

BGI graphics driver

Constant value

DETECT - 0(REQUESTS AUTO DETECTION)

CGA – 1

MCGA – 2

EGA – 3

EGA64 – 4

EGAMONO – 5

IBM8514 – 6

ERCMONO – 7

ATT400 – 8

VGA – 9

PC3270 – 10

In c, graphics can be implemented easily. The header file Ghraphics.h and library file Graphics.lib contains definitions and explanations of all the constants and functions we need.

Four VGA modes are available:

640*200(16 color)

640 * 350(16 color)

640 * 480(16 color)

800*600 with 256 colors

08. BIBLIOGRAPHY

- **Kanetkar Yeswant.** Let us C. 5th edition, BPB publications,2005
- **Silberschatz Abraham (bell labs) & Galvin Peter Baer (Corporate technologies, inc.),** operating system concepts, 5th edition.