**CHAPTER 1**

* 1. **INTRODUCTION**

# CPU scheduling is the basis of multi programmed operating systems. By switching the CPU among processes, the operating system can make the computer more productive. This project introduce basic CPU-scheduling concepts and present several CPU-scheduling algorithms and also consider the problem of selecting an algorithm for a particular system.

# The success of CPU scheduling depends on an observed property of processes. Process execution consists of a cycle of CPU execution and I/0 wait. Processes alternate between these two states. Process execution begins with a CPU burst. That is followed by an I/O burst, which is followed by another CPU burst, then another I/0 burst, and so on. Eventually, the final CPU burst ends with a system request to terminate execution.

# 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 a process from the processes in memory that are ready to execute and allocates the CPU to that process.

CPU scheduling decision may takes place under the following circumstances:

1.When a process switches from the running state to waiting state.

2.When a process switches from the running state to the ready state.

3. When a process switches from the waiting state to the ready state.

4. When a process terminates.

* 1. **PROBLEM STATEMENT**

# In a single-processor system, only one process can run at a time; any others must wait until the CPU is free and can be rescheduled. The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization. The idea is relatively simple. A process is executed until it must wait, typically for the completion of some I/O request. In a simple computer system, the CPU then just sits idle. All this waiting time is wasted; no useful work is accomplished. With multiprogramming, here try to use this time productively.

# Thus the project overcomes this problem using scheduling algorithms by scheduling the processes. Several processes are kept in memory at one time. 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 can take over use of the CPU. Scheduling of this kind is a fundamental operating-system function.

**CHAPTER 2**

**2.1 FLOW CHART**

**Start**

# 

# 

**List of scheduling** **algorithms**

**1 1**1 **1**

**First Come First Serve**

**Shortest Job First**

**2** **2**

**3**

# 

**select any one algo**

# 

**Round Robin**

**Input**

**1.Number of processes**

**2.Arrival Time**

**3.Burst Time**

**Input**

**1.Number of processes**

**2.Arrival Time**

**3.Burst Time**

**Input**

**1.Number of processes**

**2.Arrival Time**

**3.Burst Time**

**4.Time slice**

# 

**Output**

1. **Processe**
2. **Burst Time**
3. **Waiting Time**
4. **Turnarround Time**
5. **Avg.Turnarround Time**
6. **Avg.Waiting Time**

**Exit**

**2.2 ALGORITHM**

**Input:**

We take process details . In these we take process arrival time,burst time,time slice and number of processes.

**Output:**

# When user gives the input that is process details and after selecting the type of algorithm for scheduling, then the output contains the waiting time, turnaround time for each process and also the average waiting and turnaround time for selected type of scheduling algorithm.

**Algorithm for add student FCFS :**

**Step.1**- Input the processes along with their burst time (bt).

**Step.2**- Find waiting time (wt) for all processes.

**Step.3**- As first process that comes need not to wait so

waiting time for process 1 will be 0 i.e. wt[0] = 0.

**Step.4**- Find **waiting time** for all other processes i.e. for process i ->

wt[i] = bt[i-1] + wt[i-1] .

**Step.5**- Find **turnaround time** = waiting\_time + burst\_time

for all processes.

**Step.6**- Find **average waiting time** =

total\_waiting\_time / no\_of\_processes.

**Step.7**- Similarly, find **average turnaround time** =

total\_turn\_around\_time / no\_of\_processes.

**Algorithm for add student SJF :**

**Step.1**- Input the processes along with their burst time (bt).

**Step.2**- Sort all the processes according to their burst time(ascending)

**Step.3**- Find waiting time (wt) for all processes.

**Step.4**- As first process that comes need not to wait so

waiting time for process 1 will be 0 i.e. wt[0] = 0.

**Step.5**- Find **waiting time** for all other processes i.e. for process i ->

* + - * 1. wt[i] = bt[i-1] + wt[i-1] .

**Step.6**- Find **turnaround time** = waiting\_time + burst\_time

for all processes.

**Step.7**- Find **average waiting time** =

total\_waiting\_time / no\_of\_processes.

**Step.8**-Similarly, find **average turnaround time** =

total\_turn\_around\_time / no\_of\_processes.

**Algorithm for add student RR :**

**Step.1**- Declare arrival[], burst[], wait[], turn[] arrays and initialize them. Also declare a

time variable and initialize it to zero. To sustain the original burst array create another array (temp\_burst[]) and copy all the values of burst array in it.

**Step.2**- To keep a check we create another array of bool type which keeps the record of

whether aprocess is completed or not. we also need to maintain a queue array which contains the process indices (initially the array is filled with 0).

**Step.3**- Now we increment the timer variable until the first process arrives and when it

does, we add the process index to the queue array

**Step.4**- Now we execute the first process until the time quanta and during that time quanta,

we check whether any other process has arrived or not and if it has then we add the index in the queue.

**Step.5**- Now, after doing the above steps if a process has finished, we store its exit time and

execute the next process in the queue array. Else, we move the currently executed process at the end of the queue (by calling another fxn. queueMaintainence()) when the time slice expires.

**Step.6**- The above steps are then repeated until all the processes have been completely

executed. If a scenario arises where there are some processes left but they have not arrived yet, then we shall wait and the CPU will remain idle during this interval.

**CHAPTER 3**

**3.1 DATA STRUCTURES**

**1.STRUCTURE :**

Structure is used to store the process details

**2. LINEAR QUEUE :**

Linear queue is used for FCFS algorithm.

**3. CIRCULAR QUEUE :**

Circular queue is used for RR algorithm.

**4. PRIORITY QUEUE :**

Priority queue is used for SJF algorithm.

**Chapter 4**

**4.1 Hardware Requirement**

• Processor :- 8086 Family onwords

• RAM :- 128 MB &above

• HDD :- 10 GB &above

**4.2 Software Requirements**

• Platform : Windows 11

• Programming Language :- C

• Compiler :- gcc (GNU C Compiler)

• IDE : Codeblocks

**Chapter 5**

**5.1 SYSTEM DEFINED FUNCTIONS**

**1. gotoxy():**

gotoxy() function is used to print the statement at appropriate position on the screen.

The parameters passed to the gotoxy() function are colunm number and row number.

**5.2 User defined functions**

**1. void display\_res( )**

These function is used for displaying the result of the process.

**2. void completion\_time ( )**

These function is used to calculate the completion time of the process.

**3. void Waiting\_time ( )**

with help of these functions we calculate the waiting time of the process.

**4. void sort\_with\_BT ( )**

These function is used to sort the process according to their burst time.

**5. void sort\_with\_AT( )**

These function is used to sort the process according to their arrival time.

**6. void Avg\_Turnarraound\_Time ( )**

These function is used to calculate the average turnaround time.

**7. void Avg\_Waiting\_Time ( )**

These function is used to calculate the average waiting time.

**8. void FCFS ( )**

These function is used to schedule the processes with FCFS algorithm.

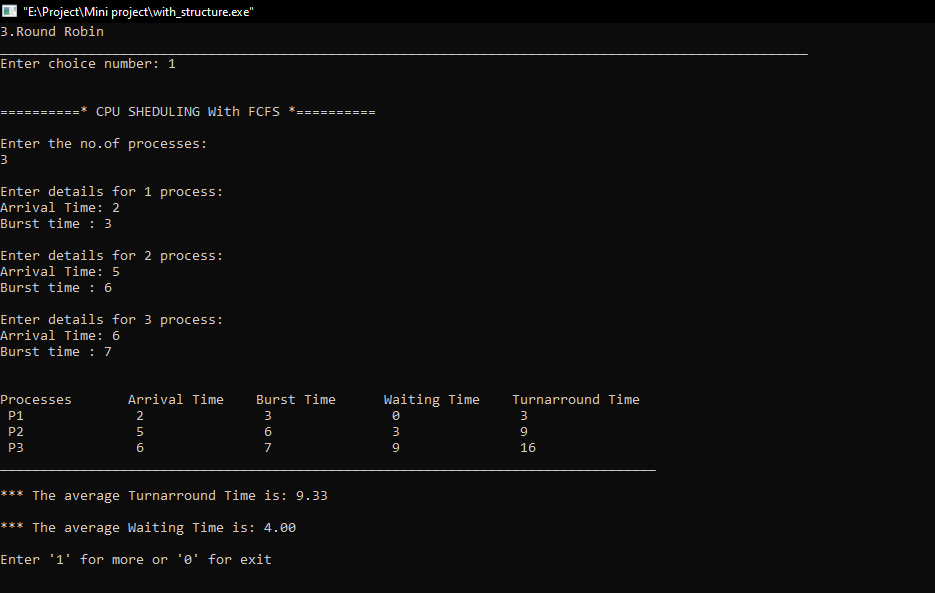
**9. void SJF ( )**

These function is used to schedule the processes with SJF algorithm..

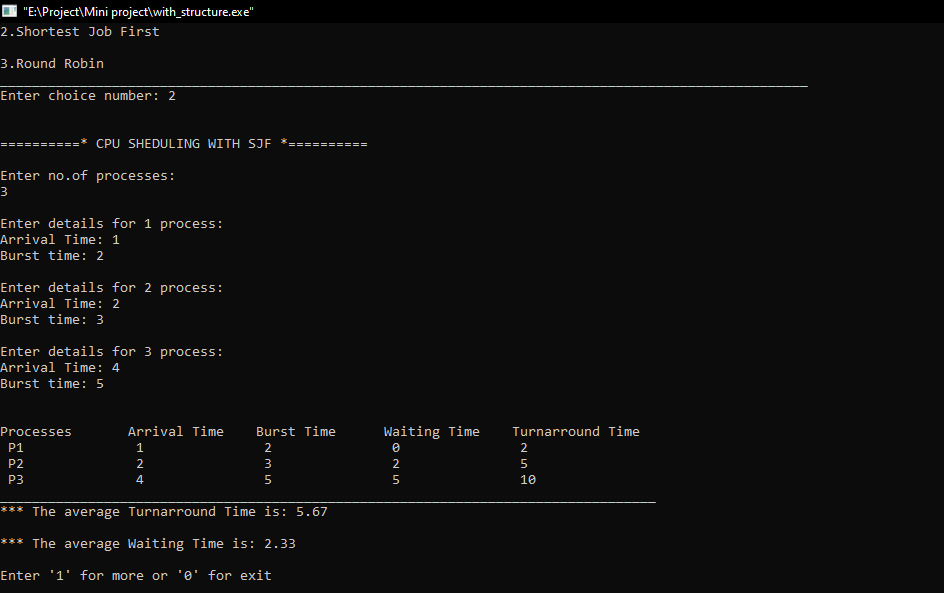
**10. void RR ( )**

These function is used to schedule the processes with RR algorithm.

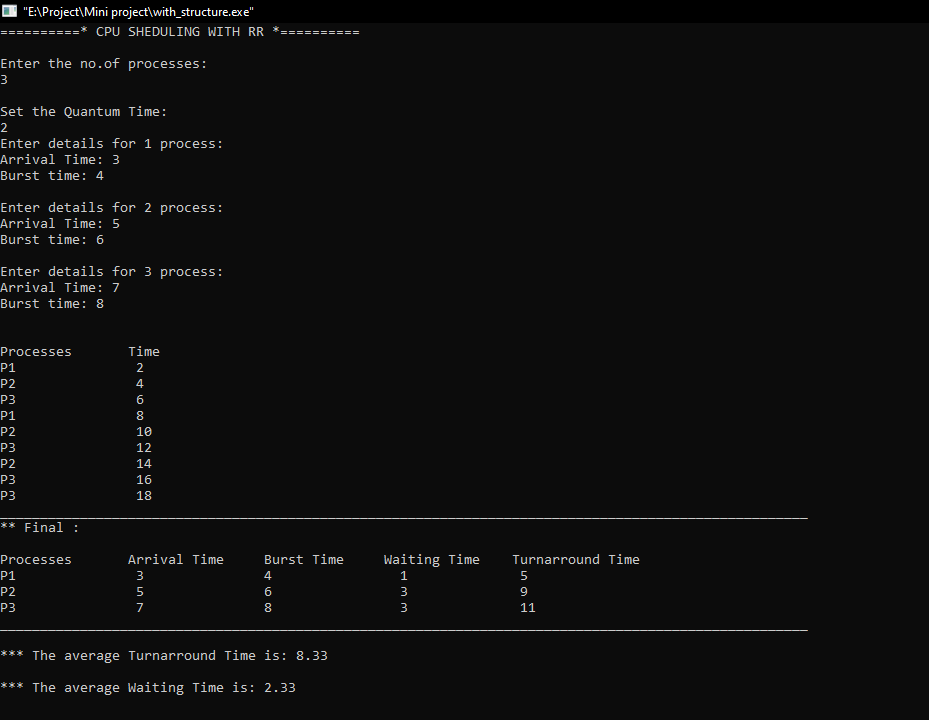
**5.3 USER MANUALS**

****

The above picture shows the output when we select the FCFS algorithm for scheduling.

****

The above picture shows the output when we select the SJF algorithm for scheduling.

****

The above picture shows the output when we select the RR algorithm for scheduling.

**5.4 conclusions**

The treatment of shortest process in SJF Scheduling tends to result in increased waiting time for long processes. And the long process will never get served, though it produces minimum average waiting time and average turnaround time. It is recommended that any kind of simulation for any CPU scheduling algorithm has limited accuracy.

**REFERENCES**

**Reference book**

1. [1] Silberschatz, A. P.B. Galvin and G. Gagne (2012), Operating System Concepts, 8th

edition, Wiley India,

2. The complete reference C /4th edition - herbert schildt

**Web reference**

1. https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/

2. https://www.tutorialspoint.com/operating\_system/os\_process\_scheduling.htm