TABLE OF CONTENTS

CONTENTS

FCFS based CPU scheduler

Introduction:

CPU scheduling:

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold (in waiting time) due to unavailability of any resource like I/O, thereby making full use of CPU. It is the basis of multiprogrammed operating system. By switching the CPU among processes, the operating system can make the computer more productive.

- 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.
  - 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 work is accomplished.
- With multiprogramming, we try to use this time productively.
- Several processes are kept in memory at one time.

- When one process has to wait, the operating system takes the CPU away from that processor and gives the CPU to another process and the pattern continues.
- ❖ CPU burst is when the process is being executed in the CPU i.e., the time taken by the CPU to complete an execution of a process is **Burst Time.**

CPU Scheduler:

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 are ready to execute and allocate the CPU to that process.

Dispatcher:

The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler.

The time taken for the dispatcher to stop one process and start another running is known as the ***dispatch latency*.**

Types of CPU Scheduling:

CPU-scheduling decisions may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state.
2. When a process switches from the running state to the ready state for example, when an interrupt occurs.
3. When a process switches from the waiting state to the ready state for example, at completion of I/O.
4. When a process terminates.

For situation 1 and 4, there is a choice in terms of scheduling. A new process (if one exits in the ready queue) must be selected for executing. However, there is a choice for situation 2 and 3.

When scheduling takes place only under circumstances 1 and 4, we say that the scheduling scheme is **nonpreemptive** or cooperative; otherwise, it is **preemptive.**

Non-Preemptive Scheduling:

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 time.

Preemptive Scheduling:

In this type of scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority has finished its execution.

Scheduling Algorithms:

To decide which process to execute first and which process to execute last to achieve maximum CPU utilization, computer scientists have defined some algorithms, they are:

1. First-Come, First-Serve (FCFS) Scheduling.
2. Shortest job First (SJF) Scheduling.
3. Priority Scheduling.
4. Round Robin (RR) Scheduling.

Scheduling criteria:

CPU utilization:

We want the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, is should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily loaded system).

Throughput:

If the CPU is busy executing processes, then work is being done. **One measure of work is the number of processes that are completed per unit time, called throughput.**

Turnaround time:

From the point of view of a particular process, the important criterion is how long it takes to execute that process. The interval from the time of submission of a process to the time of completion is turnaround time**. Turnaround time is the sum of the periods spent waiting to get into the memory, waiting in the ready queue, executing the CPU, and doing I/O.**

Waiting time:

The CPU scheduling algorithm does not affect the amount of time during which a process executes or does I/O; if affects only the amount of time that a process spends waiting in the ready queue. **Waiting time is the sum of the periods spent in the ready queue.**

Response Time:

In an interactive system, turnaround time may not be the best criterion. Often, a process can produce some output fairly early and can continue computing new results while previous results are being output to the user. **Thus, another measure is the time from the submission of a request until the first response is produced. This measure, called response time, is the time takes to start responding, not the time it takes to output the response**. The turnaround time is generally limited by the speed of the output device.

Arrival Time:

Time at which the process arrives in the ready queue.

Burst Time:

It is the time required by a process for CPU execution.

Completion Time:

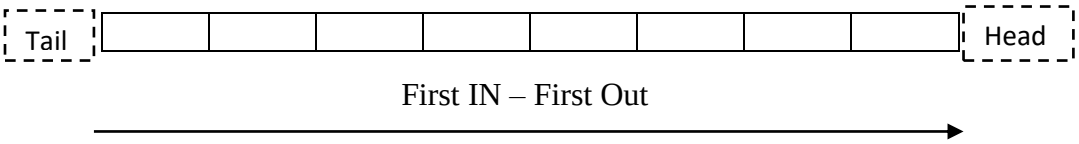
It is the time at which the process completes its execution.

FIRST-COME, FIRST-SERVE SCHEDULING ALGORITHM:

FCFS is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival.

Characteristics:

- FCFS is by far the simplest CPU-scheduling algorithm.
- The process that requests the CPU first is allocated the CPU first.
- The implementation of the GCGS policy is easily managed with a **FIFO queue.**
- When a process enters the ready queue, its Process Control Block is linked onto the tail of the queue.



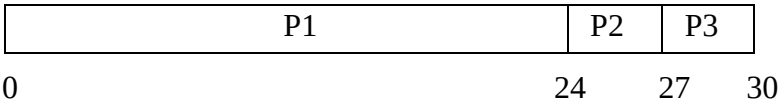
- When the CPU is free, it is allocated to the process at the head of the queue.

- **The average waiting time under the FCFS policy, however, is often quite long.**

Consider the following set of processes that arrives at time 0.

Process	Burst Time (ms)
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:



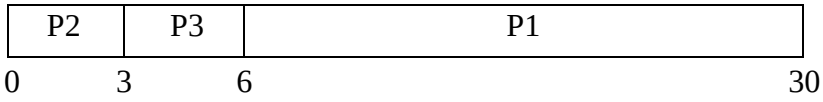
Waiting time for P1= 0 ms

Waiting time for P2= 24 ms

Waiting time for P3= 27 ms

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

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



Waiting time for P1= 6 ms

Waiting time for P2= 0 ms

Waiting time for P3= 3 ms

Average waiting time= (6+0+3)/3 = 3ms.

This reduction is substantial. Thus, the average waiting time under FCFS policy is generally not minimal and may vary substantially if the process's CPU burst times vary greatly.

The FCFS scheduling algorithm is nonpreemptive

- 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.
- The FCFS algorithm is thus particularly troublesome for time-sharing systems, where it is important that each user get a share of the CPU at regular intervals.
- It would be disastrous to allow one process to keep the CPU for an extended period.

Problem Statement: Implement FCFS based CPU Scheduler

Modules with description of their functionalities along with logic:

#- It is a preprocessor command, used to process the functionality prior to other statements in the program. In a computer system, '#' is the symbol that has the highest priority over any other symbol so for the execution of some prioritized statements (library functions or defining macros) we use #.

#include- The #**include** preprocessor directive is **used** to paste code of given file into current file. If **included** file is not found, compiler renders error. By the use of #**include** directive, we provide information to the preprocessor where to look for the header files.

#include<stdio.h>- Standard input-output header is a header file used to perform input and output operations in C like scanf() and printf().

#include<conio.h>- Console input-output header is a header file, performs console input and console output operations like clrscr() to clear the screen and getch() to get the character from the keyboard.

Int main()- main() is a mandatory function in C programs. It defines the entry point of the programs. int is the return type of the function, main() returns to the Operating System. main() returns 0 on success and 1 on failure.

{- defining the start of the main function.

Int- I**nt**, short for "**integer**," is a fundamental variable type built into the compiler and **used** to define numeric variables holding whole numbers.

n- integer data-type, used to take the number of processes.

i, j- looping variables.

temp- temporary variable, used in sorting.

P[30]- integer array to take Process IDs, to maximum of 30.
AT[30]- Arrival Time array,
BT[30]- Burst Time array,
CT[30]- Completion Time array,
TAT[30]- Turn-around Time array,
WT[30]- Waiting Time array.
} All are integer type, used to store the respective data.

float- fundamental variable, used for storing values containing decimal places.

sTAT- sum of the all the turn-around times, initialized to zero.
sWT- sum of all waiting times, initialized to zero.
aTAT- average of turn-around time
aWT- average to waiting time
} All are of float data-type.

clrscr()- console input-output, used to clear the screen.

printf()- this function is used to print onto the output screen.

\n- to print the statement mentioned in the printf(…) function onto the next line.
\t- prints out a tab which is an undefinable amount of space that aligns the next section of output to a horizontal tab on the screen.

scanf()- reads formatted input from the standard input such as keyboards.

&- ampersand(&) allows us to pass address of variable number which is the place in memory Where we store the information that scanf read.

%d- format specifier for an integer value, used in the formatted output function printf() to output any value of the type integer and used to take input of the type integer through scanf() function.

%.3f- format specifier for an integer value in decimals used in the formatted output function printf() to output any value of the type integer in decimals and used to take input of the type integer through scanf() function. .3 here to display decimal values only upto 3 decimal places.

getch()- it prompts a user to press a character.

Program logic:

- The header included is <stdio.h> and <conio.h>.
- The main () function, it is of integer type so it returns a value.
- ‘n’ is an integer variable used to take the number of processes to be performed by the processor, ‘i and j’ are used for looping purpose, ‘temp’ is an integer variable used while sorting, P[30] is an integer

array collection(0-29) of Process ID, AT[30] is an integer array collection(0-29) of arrival times of the processes in the ready queue, BT[30] is an integer array collection(0-29) of burst times of the processes, CT[30] is an integer array collection(0-29) of completion times of the processes, WT[30] is an integer array collection(0-29) of waiting times of the processes in the ready queue, TAT[30] is an integer array collection(0-29) of turn-around times of the processes.

' sWT, sTAT, aWT, aTAT are float variable type. sWT and sTAT are initialized to zero.

' Read the number of processes to be performed by the processor.

' Run a for loop, initializing the looping variable i=0, and running the loop till i<n incrementing on each step and read the process Id , arrival time and burst time.

' Sort PID, AT and BT with respect to arrival time: Run two for loops-

First for loop, initialize the looping variable i=0, run till i<n incrementing on each i.

Second for loop, initialize the looping variable j=i+1, run till i<n incrementing on each j.

' if statement AT[i]<AT[j] sorts the elements in increasing order, compare arrival times for two different looping variables, if the 'if-statement is true then consider the following code mentioned inside the if-statement. If it is false, skip that part.

' Consider the temporary variable 'temp', assign temp=AT[i] and let AT[i] be equal to At[j]. Therefore At[j] will be equal to temp. similarly sorting PIDs and BTs.

' Completion time of a process is got by subtracting the completion time of the previous process with the burst time of the present process. Therefore the completion time of the process at zeroth index is equal to the burst time of the process at zeroth index. So run a for loop, initializing i=1, upto i<n incrementing, to calculate the completion of the processes from the oneth index.

' Run another for loop, initializing i=0, upto i<n incrementing to calculate the turn-around time and waiting time.

Turn-around time is got by subtracting completion time of a process with arrival time of that process. Take the sum of all the TAT[i]. Waiting time is got by subtracting turn-around time of a process with burst time of that process. Take the sum of all the WT[i].

' Calculating the average turn-around time by dividing the sum of turn-around time and the total number of processes processed by the processor, similarly calculating average waiting time.

' Displaying the PIDs with their respected arrival time, burst time, calculated completion time, turn-around time and waiting time.

' Printing the average of turn-around time and waitng time to three decimal places.

' Since the main() funstion is integer type return 0.

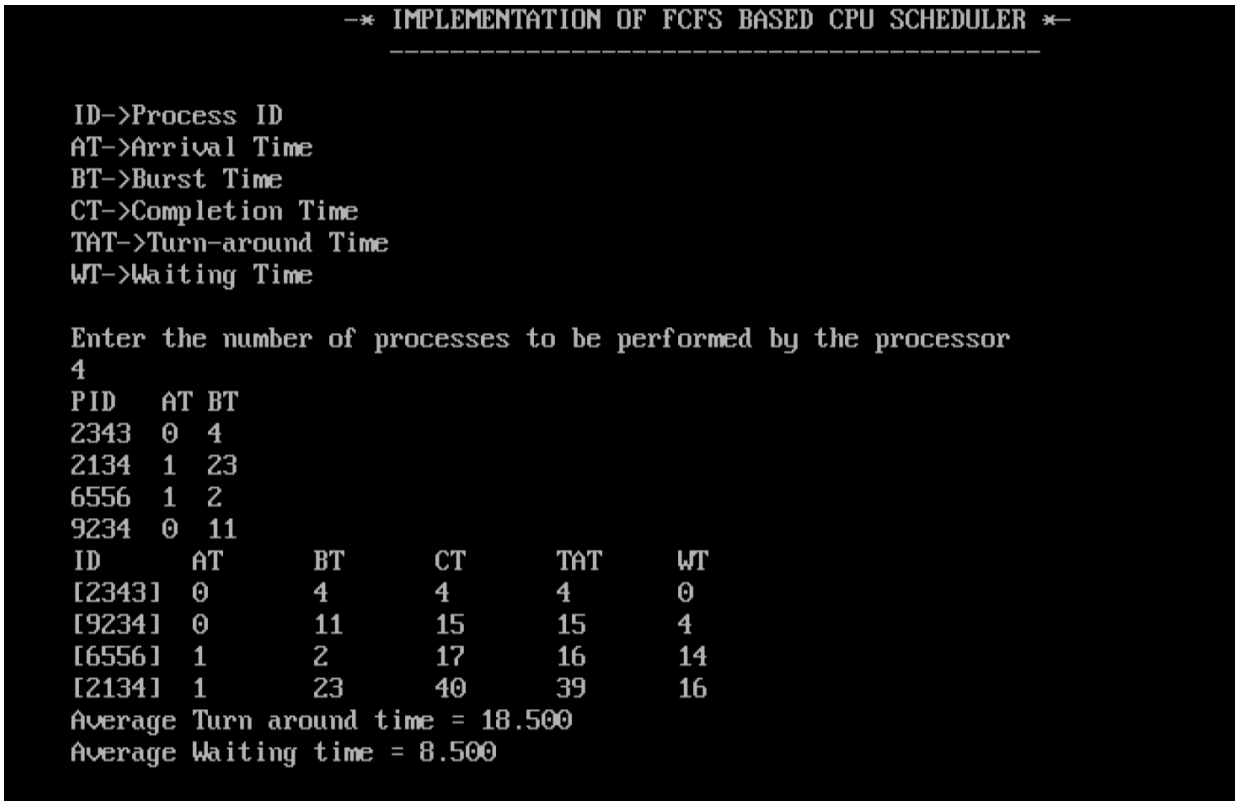
IMPLEMENTATION:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int n, i, j, temp, P[30];
    int AT[30], BT[30], CT[30], WT[30],  TAT[30];
    float sTAT=0, sWT=0, aTAT, aWT;
    clrscr();
    printf("               -* IMPLEMENTATION OF FCFS BASED CPU SCHEDULER *-\n");
    printf("                 ----------------------------------------\n");
    printf("\n");
    printf("ID->Process ID\nAT->Arrival Time\nBT->Burst Time\nCT->Completion Time\nTAT->Turn-around
Time\nWT->Waiting Time\n");
    printf("\n");
    printf("Enter the number of processes to be performed by the processor\n");
    scanf("%d", &n);
    printf("PID   AT BT\n");
    for(i=0; i<n; i++)
   {
        scanf("%d", &P[i]);
        scanf("%d", &AT[i]);
        scanf("%d", &BT[i]);
   }

   for(i=0; i<n; i++)
   {
       for(j=i+1; j<n; j++)
       {
            if(AT[i]>AT[j])
            {
                 temp=AT[i];
                 AT[i]=AT[j];
                 AT[j]=temp;
                 temp=P[i];
                 P[i]=P[j];
                 P[j]=temp;
                 temp=BT[i];
                 BT[i]=BT[j];
                 BT[j]=temp;
            }
```

```
        }
    }

    CT[0]=BT[0];
    for(i=1; i<n; i++)
    {
        CT[i]=CT[i-1]+BT[i];
    }

    for(i=0;  i<n; i++)
    {
         TAT[i]=CT[i]-AT[i];
         sTAT+=TAT[i];
        WT[i]=TAT[i]-BT[i];
         sWT+=WT[i];
    }

    aTAT=sTAT/n;
    aWT=sWT/n;

    printf("ID\tAT\tBT\tCT\tTAT\tWT\n");
    for(i=0; i<n; i++)
    {
        printf("[%d]\t%d\t%d\t%d\t%d\t%d\n", P[i], AT[i], BT[i], CT[i], TAT[i], WT[i]);
    }
    printf("Average Turn around time = %.3f\n", aTAT);
    printf("Average Waiting time = %.3f\n", aWT);

    getch();
    return 0;
}
```

OUTPUTS:

```
                    -* IMPLEMENTATION OF FCFS BASED CPU SCHEDULER *-
                       ------------------------------------------


ID->Process ID
AT->Arrival Time
BT->Burst Time
CT->Completion Time
TAT->Turn-around Time
WT->Waiting Time

Enter the number of processes to be performed by the processor
3
PID   AT BT
2123  0  3
2313  1  7
7365  2  22
ID      AT      BT      CT      TAT     WT
[2123]  0       3       3       3       0
[2313]  1       7       10      9       2
[7365]  2       22      32      30      8
Average Turn around time = 14.000
Average Waiting time = 3.333
```


-* IMPLEMENTATION OF FCFS BASED CPU SCHEDULER *-
ID->Process ID
AT->Arrival Time
BT->Burst Time
CT->Completion Time
TAT->Turn-around Time
WT->Waiting Time
Enter the number of processes to be performed by the processor
4
PID AT BT
2343 0 4
2134 1 23
6556 1 2
9234 0 11
ID AT BT CT TAT WT
[2343] 0 4 4 4 0
[9234] 0 11 15 15 4
[6556] 1 2 17 16 14
[2134] 1 23 40 39 16
Average Turn around time = 18.500
Average Waiting time = 8.500

Advantages of FCFS

Here, are pros/benefits of using scheduling algorithm:

- The simplest form of a CPU scheduling algorithm.
- Easy to program.
- First come first served.

Disadvantages of FCFS

Here, are cons/drawbacks of using FCFS scheduling algorithm:

- It is a Non-preemptive CPU scheduling algorithm, so after the process has been allocated to the CPU, it will never release the CPU until it finishes executing.
- The Average Waiting Time is high.
- Short processes that are at the back of the queue have to wait for the long process at the front to finish.
- Not an ideal technique for time-sharing systems.
- Because of its simplicity, FCFS is not very efficient.