

CPU scheduling simulation



Write a program to simulate CPU Scheduling Algorithms: FCFS, SJF (Preemptive), Priority (Non-Preemptive) and Round Robin (Preemptive)

```
#include<iostream>
using namespace std;

struct fcfs
{
    int burst, arrival, id, completion, waiting, turnaround, response;
};

fcfs meh[30];

class FCFS
{
public:
    int n;
    void fcfsIn(){
        cout<<"\nEnter number of processes: ";
        cin>>n;
        for(int i = 0; i < n; i++){
            cout<<"\nEnter arrival time of P"<<i<<": ";
            cin>>meh[i].arrival;
            cout<<"\nEnter burst time of P"<<i<<": ";
            cin>>meh[i].burst;
            meh[i].id = i;
        }
        cout<<"\n | Arrival | Burst\n";
        for(int j = 0; j < n; j++) {
            cout<<"P"<<j<<" | "<<meh[j].arrival<<" | "<<meh[j].burst<<"\n";
        }
    }

    void process() {
        cout<<"\nSequence of processes is: ";
        int currentTime = 0;

        for(int i = 0; i < n; i++){
            if(currentTime < meh[i].arrival){
                while(currentTime < meh[i].arrival){
                    cout<<" NULL ";
                    currentTime++;
                }
            }
        }
    }
}
```

```

meh[i].response = currentTime - meh[i].arrival;
cout<<"P"<<meh[i].id<< " ";
currentTime += meh[i].burst;
meh[i].completion = currentTime;
meh[i].turnaround = meh[i].completion - meh[i].arrival;
meh[i].waiting = meh[i].turnaround - meh[i].burst;
}
}

void displayMetrics()
{
    double totalWaiting = 0, totalTurnaround = 0, totalCompletion = 0;

    cout<<"\n\n | Completion time | Waiting time | Turnaround time | Response time\n";
    for(int j = 0; j < n; j++) {
        totalWaiting += meh[j].waiting;
        totalTurnaround += meh[j].turnaround;
        totalCompletion += meh[j].completion;
        cout<<"P"<<j<< " | "<<meh[j].completion
            <<"      | "<<meh[j].waiting
            <<"      | "<<meh[j].turnaround
            <<"      | "<<meh[j].response<<"\n";
    }

    cout<<"\nAverage completion time: "<<totalCompletion/n;
    cout<<"\nAverage waiting time: "<<totalWaiting/n;
    cout<<"\nAverage turnaround time: "<<totalTurnaround/n;
}
};

int main()
{
    FCFS obj;
    obj.fcfsIn();
    obj.process();
    obj.displayMetrics();
    return 0;
}

```

explain code as well as give input for running it

Sure — let's go step by step 

Purpose of the Program

This program **simulates the FCFS (First Come First Serve)** CPU scheduling algorithm, one of the simplest scheduling algorithms used by an operating system to decide which process runs next on the CPU.

Later, you can extend it for **SJF, Priority, and Round Robin**, but currently, this code is for **FCFS only**.

Understanding the Code

1 Structure Definition

```
cpp

struct fcfs
{
    int burst, arrival, id, completion, waiting, turnaround, response;
};
fcfs meh[30];
```

- This structure represents each **process**.
 - **burst**: how long the process takes (CPU time needed)
 - **arrival**: when the process arrives
 - **id**: process number (P0, P1, P2, etc.)
 - **completion**: time when process finishes execution
 - **waiting**: time process waited before getting CPU
 - **turnaround**: total time from arrival to completion
 - **response**: time from arrival to first CPU response
-

2 Class FCFS

This class handles **input**, **processing**, and **output**.

3 Input Function

```
cpp

void fcfsIn() {
    cout<<"\nEnter number of processes: ";
    cin>>n;
    for(int i = 0; i < n; i++){
        cout<<"\nEnter arrival time of P"<<i<<": ";
        cin>>meh[i].arrival;
        cout<<"\nEnter burst time of P"<<i<<": ";
        cin>>meh[i].burst;
        meh[i].id = i;
    }
}
```

- Takes the number of processes **n**.
 - For each process, takes **arrival time** and **burst time**.
 - Stores values into the **meh** array.
-

4 Process Scheduling Logic

```
cpp

void process() {
    cout<<"\nSequence of processes is: ";
    int currentTime = 0;

    for(int i = 0; i < n; i++){
        if(currentTime < meh[i].arrival){
            while(currentTime < meh[i].arrival){
                cout<<" NULL ";
                currentTime++;
            }
        }

        meh[i].response = currentTime - meh[i].arrival;
        cout<<"P"<<meh[i].id<<" ";
        currentTime += meh[i].burst;
        meh[i].completion = currentTime;
        meh[i].turnaround = meh[i].completion - meh[i].arrival;
        meh[i].waiting = meh[i].turnaround - meh[i].burst;
    }
}
```

✓ Explanation:

- `currentTime` keeps track of CPU time.
- If a process hasn't arrived yet, CPU stays **idle** (**NULL** printed).
- Once process arrives, it:
 - Calculates **response time** (`currentTime - arrival`)
 - Executes process (`currentTime += burst`)
 - Calculates:
 - `completion = currentTime`
 - `turnaround = completion - arrival`
 - `waiting = turnaround - burst`

5 Displaying Results

```
cpp

void displayMetrics()
{
    double totalWaiting = 0, totalTurnaround = 0, totalCompletion = 0;
    cout<<"\n\n | Completion time | Waiting time | Turnaround time | Response time\n";
    for(int j = 0; j < n; j++) {
        totalWaiting += meh[j].waiting;
        totalTurnaround += meh[j].turnaround;
        totalCompletion += meh[j].completion;
        cout<<"P"<<j<<" | "<<meh[j].completion
             <<"           | "<<meh[j].waiting
             <<"           | "<<meh[j].turnaround
             <<"           | "<<meh[j].response<<"\n";
    }
    cout<<"\nAverage completion time: "<<totalCompletion/n;
    cout<<"\nAverage waiting time: "<<totalWaiting/n;
```

```

    cout<<"\nAverage turnaround time: "<<totalTurnaround/n;
}

```

✓ Explanation:

- Prints table of all times for each process.
- Computes averages for performance analysis.

6 Main Function

```

cpp

int main()
{
    FCFS obj;
    obj.fcfsIn();
    obj.process();
    obj.displayMetrics();
    return 0;
}

```

This is the driver function:

- Creates an **FCFS** object.
- Calls input, process, and display methods in order.

Sample Input

When you run the program, enter the following sample values:

```

yaml

Enter number of processes: 4
Enter arrival time of P0: 0
Enter burst time of P0: 5
Enter arrival time of P1: 1
Enter burst time of P1: 3
Enter arrival time of P2: 2
Enter burst time of P2: 8
Enter arrival time of P3: 3
Enter burst time of P3: 6

```

Expected Output

```

pgsql

| Arrival | Burst |
P0| 0      | 5   |
P1| 1      | 3   |
P2| 2      | 8   |
P3| 3      | 6   |

```

Sequence of processes is: P0 P1 P2 P3

	Completion time	Waiting time	Turnaround time	Response time
P0	5	0	5	0
P1	8	4	7	4
P2	16	6	14	6
P3	22	13	19	13

Average completion time: 12.75

Average waiting time: 5.75

Average turnaround time: 11.25

🧠 Concept Recap

Metric	Formula
Completion Time	Time when process finishes
Turnaround Time	Completion - Arrival
Waiting Time	Turnaround - Burst
Response Time	Time from arrival until first execution
CPU Idle	When <code>currentTime < arrival</code>

Would you like me to now extend this same program to include **SJF (Preemptive)**, **Priority (Non-Preemptive)**, and **Round Robin (Preemptive)** — all in one combined code?



now give viva 25 question along with solution for the same code

Perfect 👍 — here are **25 viva questions with clear answers** based on your **CPU Scheduling (FCFS)** simulation code.

These are the **same type of questions** you can expect in lab viva or practical exam.

🧩 Viva Questions and Answers (for FCFS Scheduling Code)

1. What does CPU scheduling mean?

Answer:

CPU scheduling is the process of deciding which process in the ready queue should be executed by the CPU next.

2. What is FCFS scheduling?

Answer:

FCFS (First Come First Serve) is the simplest CPU scheduling algorithm where the process that arrives first is executed first.

3. Is FCFS preemptive or non-preemptive?

Answer:

FCFS is a **non-preemptive** scheduling algorithm — once a process starts execution, it runs until completion.

4. What are the key parameters used in CPU scheduling?

Answer:

- Arrival Time
 - Burst Time
 - Completion Time
 - Waiting Time
 - Turnaround Time
 - Response Time
-

5. What is Arrival Time?

Answer:

Arrival Time is the time at which a process enters the ready queue.

6. What is Burst Time?

Answer:

Burst Time is the amount of CPU time required by a process to complete its execution.

7. What is Completion Time?

Answer:

Completion Time is the time when the process finishes its execution.

8. What is Turnaround Time?

Answer:

Turnaround Time = Completion Time - Arrival Time

It is the total time taken from arrival to completion.

9. What is Waiting Time?

Answer:

Waiting Time = Turnaround Time - Burst Time

It is the total time the process spends waiting in the ready queue.

10. What is Response Time?

Answer:

Response Time = Time from arrival to the first CPU allocation for that process.

11. How does FCFS handle CPU idle time?

Answer:

If the next process has not yet arrived, the CPU remains **idle** until the process arrives. In this code, it prints "NULL" during idle time.

12. What is the time complexity of FCFS scheduling?

Answer:

$O(n)$, since each process is visited once in sequence.

13. Does FCFS cause starvation?

Answer:

No, FCFS does not cause starvation because every process is executed in the order of arrival.

14. What is convoy effect in FCFS?

Answer:

The **convoy effect** occurs when a long process blocks the CPU and causes all shorter processes to wait behind it.

15. How are processes represented in your code?

Answer:

Processes are represented using a **structure** (`struct fcfs`) that stores details like burst, arrival, waiting, turnaround, and completion time.

16. How does your code calculate average waiting time?

Answer:

By summing up the waiting time of all processes and dividing by the total number of processes:

```
cpp  
totalWaiting / n
```

17. Why do we calculate average turnaround time?

Answer:

To measure how efficiently the CPU executes and completes all processes from arrival to completion.

18. What is the role of currentTime in the code?**Answer:**

`currentTime` keeps track of CPU time as processes execute and helps calculate completion, waiting, and turnaround times.

19. What happens if two processes arrive at the same time in FCFS?**Answer:**

If arrival times are the same, the process with the smaller **process ID** (i.e., input order) is executed first.

20. What is the advantage of FCFS scheduling?**Answer:**

- Simple to understand and implement.
 - Fair — processes are executed in order of arrival.
-

21. What are the disadvantages of FCFS?**Answer:**

- Convoy effect (short jobs wait for long ones).
 - Poor average waiting time for varied burst times.
 - Not suitable for time-sharing systems.
-

22. How would you make this program work for SJF or Round Robin?**Answer:**

By changing the selection rule:

- **SJF:** Select process with smallest remaining burst time.
 - **Round Robin:** Execute each process for a fixed time quantum in cycles.
-

23. How is the Gantt chart represented in this program?**Answer:**

The sequence printed as "`P0 P1 P2 ...`" represents the **Gantt chart** (process execution order). "`NULL`" indicates CPU idle time.

24. How can this FCFS code be improved?

Answer:

- Sort processes by arrival time before execution.
 - Add support for preemptive algorithms.
 - Display Gantt chart with timestamps.
-

25. What are the performance criteria for comparing scheduling algorithms?**Answer:**

1. CPU utilization
 2. Throughput
 3. Turnaround time
 4. Waiting time
 5. Response time
 6. Fairness
-

 **Quick Formula Summary**

Term	Formula
Completion Time	When process finishes
Turnaround Time	CT - AT
Waiting Time	TAT - BT
Response Time	Start Time - AT
CPU Utilization	(Busy Time / Total Time) × 100

Would you like me to now give **viva questions and answers for all four algorithms (FCFS, SJF, Priority, and Round Robin)** combined — so you can use them in one practical submission?