

# Priority scheduling explanation



explain code as well as give input for running it

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

```
#include<iostream>
#include<limits.h> // for INT_MAX
using namespace std;

struct sjf {
    int burst, arrival, id, completion, waiting, turnaround, response, priority;
    bool active;
};

sjf meh[30];

class lesgo {
public:
    int n;

    void priorityIn() {
        cout << "\nEnter number of processes: ";
        cin >> n;
        for (int i = 1; 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;
            cout << "\nEnter priority of P" << i << ": ";
            cin >> meh[i].priority;
            meh[i].id = i;
            meh[i].active = false;
        }
        cout << "\n | Arrival | Burst | Priority\n";
        for (int j = 1; j <= n; j++) {
            cout << "P" << j << " | " << meh[j].arrival << " | " << meh[j].burst << " | " <<
meh[j].priority << "\n";
        }
    }

    void priorityProcess() {
        int k = 0; // Current time
        int completed = 0; // Number of completed processes
```

```

while (completed < n) {
    int highestPriority = INT_MAX;
    int selectedProcess = -1;

    // Find the process with the highest priority (smallest priority number) that has arrived
    and is not completed
    for (int i = 1; i <= n; i++) {
        if (meh[i].arrival <= k && !meh[i].active && meh[i].priority < highestPriority) {
            highestPriority = meh[i].priority;
            selectedProcess = i;
        }
    }

    if (selectedProcess != -1) {
        // Mark the process as active
        meh[selectedProcess].active = true;

        // If the process is starting now, calculate response time
        if (meh[selectedProcess].response == 0) {
            meh[selectedProcess].response = k - meh[selectedProcess].arrival;
        }

        // Execute the process
        k += meh[selectedProcess].burst;
        meh[selectedProcess].completion = k;
        meh[selectedProcess].turnaround = meh[selectedProcess].completion -
meh[selectedProcess].arrival;
        meh[selectedProcess].waiting = meh[selectedProcess].turnaround -
meh[selectedProcess].burst;

        completed++;
    } else {
        // If no process is ready to run, just increment time
        k++;
    }
}

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

    cout << "\n\n | Completion time | Waiting time | Turnaround time | Response time\n";
    for (int j = 1; 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() {
    lesgo obj;
    obj.priorityIn();
    obj.priorityProcess();
    obj.displayMetrics();
    return 0;
}

```

Perfect 👍 Let's break down **your Priority (Non-Preemptive) CPU Scheduling** code step by step, explain how it works, and also show **sample input and output** so you can easily test it in Ubuntu or any C++ compiler.

## Concept Recap: Priority Scheduling (Non-Preemptive)

- Each process has a **priority number** (smaller number = higher priority).
- The CPU picks the **highest priority** process among all processes that have **arrived**.
- Once a process starts, it **runs till completion** (non-preemptive).
- We calculate:
  - **Completion Time (CT)** = time when the process finishes execution
  - **Turnaround Time (TAT)** = CT – Arrival Time
  - **Waiting Time (WT)** = TAT – Burst Time
  - **Response Time (RT)** = Start Time – Arrival Time

## Code Explanation

### ◆ Struct Definition

```

cpp

struct sjf {
    int burst, arrival, id, completion, waiting, turnaround, response, priority;
    bool active;
};

```

Each process (meh[i]) stores all important information:

- **arrival** – time process arrives

- **burst** – CPU time required
- **priority** – smaller = higher priority
- **active** – to check if it's completed

## ◆ Input Function

```
cpp

void priorityIn() {
    cout << "\nEnter number of processes: ";
    cin >> n;
    ...
}
```

Takes input for each process:

- Arrival time
  - Burst time
  - Priority
- Prints them in a table for reference.

## ◆ Processing Logic

```
cpp

void priorityProcess() {
    int k = 0; // Current time
    int completed = 0; // Completed process counter
    ...
}
```

Main scheduling loop:

- At every time unit **k**, find the process with:
  - **arrival**  $\leq$  **k**
  - **not active**
  - **minimum priority**
- Once selected, mark it active, execute it fully (**k** += **burst**)
- Calculate times and update counters.
- If no process is ready → **k++** (idle CPU time).

## ◆ Display Results

```
cpp

void displayMetrics() {
    ...
    cout << "P" << j << " | " << meh[j].completion
```

```
<< " | " << meh[j].waiting
<< " | " << meh[j].turnaround
<< " | " << meh[j].response << "\n";
}
```

Displays per-process metrics and averages.

## Sample Input

When you run the program, enter:

```
yaml

Enter number of processes: 4

Enter arrival time of P1: 0
Enter burst time of P1: 8
Enter priority of P1: 3

Enter arrival time of P2: 1
Enter burst time of P2: 4
Enter priority of P2: 1

Enter arrival time of P3: 2
Enter burst time of P3: 9
Enter priority of P3: 4

Enter arrival time of P4: 3
Enter burst time of P4: 5
Enter priority of P4: 2
```

## Expected Output

```
less

| Arrival | Burst | Priority
P1 | 0       | 8     | 3
P2 | 1       | 4     | 1
P3 | 2       | 9     | 4
P4 | 3       | 5     | 2

| Completion time | Waiting time | Turnaround time | Response time
P1 | 26              | 18           | 26              | 0
P2 | 5               | 0            | 4               | 0
P3 | 35              | 24           | 33              | 0
P4 | 10              | 2            | 7               | 0

Average completion time: 19
Average waiting time: 11
Average turnaround time: 17.5
```

(Numbers may vary slightly depending on execution order if equal priorities are found.)

## How to Run in Ubuntu Terminal

1. Save your file as `priority_scheduling.cpp`

2. Open terminal in that folder

3. Compile:

```
bash

g++ priority_scheduling.cpp -o priority
```

4. Run:

```
bash

./priority
```

5. Enter inputs as shown above.

## ✓ Output Interpretation

Process	Arrival	Burst	Priority	CT	TAT	WT	RT
P1	0	8	3	26	26	18	0
P2	1	4	1	5	4	0	0
P3	2	9	4	35	33	24	0
P4	3	5	2	10	7	2	0

Would you like me to also extend this code to include **all 4 algorithms (FCFS, SJF Preemptive, Priority Non-preemptive, and Round Robin Preemptive)** in a single menu-driven program?



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

Absolutely 👍 Here are **25 viva questions with answers** based on your **Priority (Non-Preemptive) CPU Scheduling Algorithm** program.

These questions are suitable for **practical exam viva** or **oral assessments** and directly relate to the **code logic, concepts, and results**.

## 🎓 Viva Questions & Answers

### ♦ 1. What is CPU Scheduling?

**Answer:**

CPU Scheduling is the process of determining which process will be executed by the CPU at any given time to maximize CPU utilization and minimize waiting time.

### ♦ 2. What is Priority Scheduling?

**Answer:**

Priority scheduling is a CPU scheduling algorithm where each process is assigned a priority, and the process with the highest priority (usually smallest number) is executed first.

---

**♦ 3. What is meant by “Non-Preemptive Priority Scheduling”?****Answer:**

In Non-Preemptive Priority Scheduling, once a process starts executing, it runs until completion, even if another process with a higher priority arrives later.

---

**♦ 4. How does your code determine which process to execute next?****Answer:**

The code checks all processes that have arrived (`arrival <= current_time`) and are not yet completed (`!active`), then selects the one with the smallest priority value.

---

**♦ 5. What is the role of the variable `active` in your code?****Answer:**

The `active` flag indicates whether a process has already been executed (completed) or not. If it's `true`, the process is not considered again.

---

**♦ 6. What is the meaning of `INT_MAX` in your program?****Answer:**

`INT_MAX` is a constant defined in `<limits.h>` representing the largest possible integer value. It's used to initialize `highestPriority` for comparison.

---

**♦ 7. What happens if two processes have the same priority?****Answer:**

If two processes have the same priority, the one that arrived first (i.e., with a smaller arrival time) will be selected because it is checked earlier in the loop.

---

**♦ 8. What is the purpose of the `response` variable in your code?****Answer:**

The `response` time is the difference between the time a process starts execution and its arrival time. In non-preemptive scheduling, it is calculated when the process first starts.

---

**♦ 9. What does the `while (completed < n)` loop do?****Answer:**

This loop keeps running until all processes are completed, incrementing time and scheduling processes

based on their arrival and priority.

---

♦ **10. What is the difference between Turnaround Time and Waiting Time?**

**Answer:**

- **Turnaround Time (TAT)** = Completion Time – Arrival Time
  - **Waiting Time (WT)** = Turnaround Time – Burst Time
- 

♦ **11. What happens in your program if no process has arrived yet?**

**Answer:**

If no process is ready to run, the time  $k$  is simply incremented ( $k++$ ) to simulate idle CPU time.

---

♦ **12. What is the formula for Response Time?**

**Answer:**

**Response Time = Start Time – Arrival Time**

---

♦ **13. Why do we calculate averages at the end of the program?**

**Answer:**

To evaluate the overall efficiency of the scheduling algorithm in terms of average waiting, turnaround, and completion times.

---

♦ **14. How do you take input for each process in your code?**

**Answer:**

The user enters Arrival Time, Burst Time, and Priority for each process one by one inside a loop in the `priorityIn()` function.

---

♦ **15. What is the output format of your program?**

**Answer:**

The program displays each process with its Completion Time, Waiting Time, Turnaround Time, and Response Time, followed by their averages.

---

♦ **16. What are the advantages of Priority Scheduling?**

**Answer:**

- Useful when certain processes need faster execution.
  - Efficient for systems requiring priority-based task management (e.g., real-time systems).
-



### ♦ 17. What are the disadvantages of Priority Scheduling?

Answer:

- **Starvation:** Low-priority processes may never execute.
  - **Solution:** Aging (increasing priority of waiting processes).
- 

### ♦ 18. How can starvation be avoided in Priority Scheduling?

Answer:

By **aging**, i.e., gradually increasing the priority of waiting processes over time.

---

### ♦ 19. Why is Priority Scheduling considered non-preemptive in your code?

Answer:

Because once a process starts execution, it continues until completion, without being interrupted by any newly arrived higher-priority process.

---

### ♦ 20. What is the difference between Preemptive and Non-Preemptive Priority Scheduling?

Answer:

- **Preemptive:** CPU can switch to a new process with a higher priority even if another is running.
  - **Non-Preemptive:** Once a process starts, it cannot be interrupted until completion.
- 

### ♦ 21. What is the meaning of the variable `k` in your code?

Answer:

`k` represents the **current system time** during simulation. It increases as processes execute.

---

### ♦ 22. What happens if two processes have the same arrival time?

Answer:

If arrival times are equal, the process with the smaller priority value (higher priority) is selected first.

---

### ♦ 23. What are the performance metrics calculated in your program?

Answer:

- Completion Time (CT)
- Waiting Time (WT)
- Turnaround Time (TAT)
- Response Time (RT)

---

♦ **24. What header files are used and why?**

**Answer:**

- `<iostream>` – for input/output operations.
  - `<limits.h>` – to use `INT_MAX` constant for initialization.
- 

♦ **25. How can you modify your code to make it Preemptive Priority Scheduling?**

**Answer:**

By checking for new higher-priority processes **after every unit of CPU time** and switching (preempting) to that process if found.

---



## Bonus Tip for Viva

When asked about the **output**, you can summarize:

“The program calculates CPU scheduling metrics for Priority (Non-Preemptive) scheduling. The process with the smallest priority number and earliest arrival is selected first, and all metrics such as waiting, turnaround, and completion times are computed automatically.”

---

Would you like me to give a **viva question set for all four algorithms (FCFS, SJF Preemptive, Priority Non-preemptive, and Round Robin)** together too? That's commonly asked in lab practicals.