# SJF scheduling program explanation

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

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

struct sjf{
    int id, arr, bur, comp, turn, wait, orgBur;
};
sjf meh[30];

class SJF{
public:
    int n;
    void getIn()
    {
        cout<<"Enter number of processes: ";
        cin>>n;
        for(int i = 0; i < n; i++){
            cout<<"\nEnter arrival time for process "<<i<<": ";
            cin>>meh[i].arr;
            cout<<"Enter burst time for process "<<i<<": ";
            cin>>meh[i].bur;
            meh[i].orgBur = meh[i].bur;
            meh[i].id = i;
        }
    }

    void process()
    {
        int k = 0;
        int completed = 0;
        cout<<"\nSequence of processes is: ";
        while(completed < n){
            int burst = 999;
            int idd = -1;
            for(int i = 0; i < n; i++){
                if(meh[i].arr <= k && meh[i].bur > 0){
                    if(meh[i].bur < burst){
                        burst = meh[i].bur;
                        idd = i;
                    }
                }
            }
            if(idd != -1){
                cout<<"P"<<idd<<" ";
                k++;
                meh[idd].bur--;
                if(meh[idd].bur == 0){
                    meh[idd].comp = k;
                    meh[idd].turn = meh[idd].comp - meh[idd].arr;
                    meh[idd].wait = meh[idd].turn - meh[idd].orgBur;
                    completed++;
                }
            } else{
                k++;
            }
        }
    }

    void display()
    {
        double turn = 0, comp = 0, wait = 0;
        cout<<"\n     Completed | Waiting | Turnaround |";
        for(int i = 0; i < n; i++){
            turn += meh[i].turn;
            wait += meh[i].wait;
            comp += meh[i].comp;
            cout<<"\nP"<<i<<"|  "<<meh[i].comp<<"  |  "<<meh[i].wait<<"  |  "<<meh[i].turn;
        }

        cout<<"\n\nAvg completion time: "<<comp/n;
        cout<<"\nAvg turnaround time: "<<turn/n;
        cout<<"\nAvg waiting time: "<<wait/n;
    }
};

int main()
{
    SJF ob;
    ob.getIn();
    ob.process();
    ob.display();
    return 0;
}
```

explain code as well as give input for running it

Sure 👍 Let’s break down this **C++ program for simulating the SJF (Shortest Job First - Preemptive)** CPU scheduling algorithm and then show how to run it with sample input.

---

## 🧩 Explanation of the Code

## 1. Structure Definition

```cpp
struct sjf {
    int id, arr, bur, comp, turn, wait, orgBur;
};
sjf meh[30];
```

- This defines a structure **`sjf`** for each process.
- Fields:
  - **`id`** → Process ID (like P0, P1, etc.)
  - **`arr`** → Arrival time of the process.
  - **`bur`** → Remaining burst time (changes during execution).
  - **`orgBur`** → Original burst time (for later use in calculations).
  - **`comp`** → Completion time.
  - **`turn`** → Turnaround time = Completion time – Arrival time.
  - **`wait`** → Waiting time = Turnaround time – Burst time.

---

## 2. Class SJF

The class **`SJF`** handles input, processing (scheduling), and output.

### (a) Input Function

```cpp
void getIn()
{
    cout<<"Enter number of processes: ";
    cin>>n;
    for(int i = 0; i < n; i++){
        cout<<"\nEnter arrival time for process "<<i<<": ";
        cin>>meh[i].arr;
        cout<<"Enter burst time for process "<<i<<": ";
        cin>>meh[i].bur;
        meh[i].orgBur = meh[i].bur;
        meh[i].id = i;
    }
}
```

- Takes number of processes.
- For each process, user inputs:
  - Arrival time
  - Burst time
- Initializes IDs and copies burst time to **`orgBur`**.

---

### (b) Processing Function (Preemptive SJF)

```cpp
void process()
{
    int k = 0;                 // time counter
    int completed = 0;
    cout<<"\nSequence of processes is: ";
    while(completed < n){
        int burst = 999;
        int idd = -1;
        for(int i = 0; i < n; i++){
            if(meh[i].arr <= k && meh[i].bur > 0){
                if(meh[i].bur < burst){
                    burst = meh[i].bur;
                    idd = i;
                }
            }
        }
        if(idd != -1){
            cout<<"P"<<idd<<" ";
            k++;
            meh[idd].bur--;
            if(meh[idd].bur == 0){
                meh[idd].comp = k;
                meh[idd].turn = meh[idd].comp - meh[idd].arr;
                meh[idd].wait = meh[idd].turn - meh[idd].orgBur;
                completed++;
            }
        } else{
            k++;
        }
    }
}
```

**Logic explanation:**

- The simulation runs until all processes are completed.
- At each time unit `k`, we:
    - Check which processes have **arrived** (`arr <= k`) and **not yet completed** (`bur > 0`).
    - From those, pick the one with **shortest remaining burst time**.
    - Execute that process for **1 time unit** (since this is *preemptive*).
    - If any process finishes (`bur == 0`), record:
        - Completion time (`comp`)
        - Turnaround time
        - Waiting time
- If no process is available, just increment `k` (idle CPU).

**Prints the order in which processes execute (Gantt sequence).**

---

### (c) Display Function

```cpp
void display()
{
    double turn = 0, comp = 0, wait = 0;
    cout<<"\n     Completed | Waiting | Turnaround |";
    for(int i = 0; i < n; i++){
        turn += meh[i].turn;
        wait += meh[i].wait;
        comp += meh[i].comp;
        cout<<"\nP"<<i<<"|  "<<meh[i].comp<<"  |  "<<meh[i].wait<<"  |  "<<meh[i].turn;
    }

    cout<<"\n\nAvg completion time: "<<comp/n;
    cout<<"\nAvg turnaround time: "<<turn/n;
    cout<<"\nAvg waiting time: "<<wait/n;
}
```

- Displays per-process data:
    - Completion Time
    - Waiting Time
    - Turnaround Time
- Calculates and displays averages.

---

### 3. Main Function

```
cpp

int main()
{
    SJF ob;
    ob.getIn();
    ob.process();
    ob.display();
    return 0;
}
```

Just calls the three steps — input → process → output.

---

## 🧠 Conceptual Summary

| Metric | Formula |
|---|---|
| Turnaround Time | Completion Time − Arrival Time |
| Waiting Time | Turnaround Time − Burst Time |

This program implements **Preemptive SJF** (also known as **Shortest Remaining Time First**).

---

## 💻 Sample Input

```
yaml

Enter number of processes: 4

Enter arrival time for process 0: 0
Enter burst time for process 0: 8

Enter arrival time for process 1: 1
Enter burst time for process 1: 4

Enter arrival time for process 2: 2
Enter burst time for process 2: 9

Enter arrival time for process 3: 3
Enter burst time for process 3: 5
```

---

## 📊 Expected Output

```pgsql
Sequence of processes is: P0 P1 P1 P1 P1 P3 P3 P3 P3 P3 P0 P0 P0 P0 P0 P2 P2 P2 P2 P2 P2 P2 P2 P2

     Completed | Waiting | Turnaround |
P0|  17  |  9  |  17
P1|  5  |  0  |  4
P2|  26  |  15  |  24
P3|  10  |  2  |  7

Avg completion time: 14.5
Avg turnaround time: 13
Avg waiting time: 6.5
```

---

## ⚙️ How to Run the Code in Ubuntu Terminal

1. **Save the code** in a file, say **`sjf_preemptive.cpp`**.
2. **Open terminal** and navigate to the directory where it's saved.
3. Compile the code:

   ```bash
   g++ sjf_preemptive.cpp -o sjf
   ```

4. Run the executable:

   ```bash
   ./sjf
   ```

---

Would you like me to add the **FCFS**, **Priority**, and **Round Robin** algorithms into the same program (menu-based simulation) so you can run all at once?

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

Absolutely ✅ Here are **25 viva voce questions and answers** (with explanations) based on your **Preemptive SJF (Shortest Job First)** CPU Scheduling program.

These are perfect for oral exams, lab vivas, or project presentations 👇

---

# 🎓 Viva Questions and Answers for SJF (Preemptive) CPU Scheduling

---

## 1. What is CPU Scheduling?

**Answer:**
CPU Scheduling is the process of determining which process should be assigned to the CPU at any given time. It improves CPU utilization and system responsiveness.

---

## 2. What is the objective of CPU Scheduling?

**Answer:**
To maximize CPU utilization, throughput, and minimize waiting time, turnaround time, and response time.

---

## 3. What does SJF stand for?

**Answer:**
SJF stands for **Shortest Job First**, a scheduling algorithm that selects the process with the smallest burst time for execution.

---

## 4. What is the difference between Preemptive and Non-Preemptive SJF?

**Answer:**

- **Preemptive SJF (SRTF)**: The CPU can be taken away from the current process if a new process arrives with a shorter burst time.
- **Non-Preemptive SJF**: Once a process starts, it runs until completion, regardless of new arrivals.

---

## 5. What type of scheduling algorithm is implemented in your code?

**Answer:**
The code implements **Preemptive SJF**, also called **Shortest Remaining Time First (SRTF)**.

---

## 6. What data structure is used to store process information?

**Answer:**
A **structure (`struct sjf`)** array named `meh[30]` is used to store arrival time, burst time, completion time, waiting time, etc.

---

## 7. Which fields are present in the `sjf` structure?

**Answer:**
`id`, `arr`, `bur`, `comp`, `turn`, `wait`, and `orgBur`.

---

### 8. What does `orgBur` represent in the code?

**Answer:**
It stores the **original burst time** of the process before it gets decremented during execution (used to calculate waiting time later).

---

### 9. What is the role of the variable `k` in the code?

**Answer:**
`k` acts as the **time counter**, representing the current time unit during the scheduling simulation.

---

### 10. What happens if no process is available to execute at time `k`?

**Answer:**
If no process has arrived (`arr > k`), the CPU remains **idle**, and `k` is simply incremented.

---

### 11. How is a process selected at each time unit?

**Answer:**
Among all processes that have arrived and not completed, the one with the **smallest remaining burst time** is chosen for execution.

---

### 12. What does `meh[idd].bur--` represent?

**Answer:**
It means the process `idd` is executing for one time unit, so its **remaining burst time decreases by 1**.

---

### 13. When is a process considered completed?

**Answer:**
When `meh[idd].bur` becomes `0`, the process is marked as completed, and its completion time is recorded.

---

### 14. How is Turnaround Time calculated?

**Answer:**

$$\text{Turnaround Time} = \text{Completion Time} - \text{Arrival Time}$$

---

### 15. How is Waiting Time calculated?

**Answer:**

$$\text{Waiting Time} = \text{Turnaround Time} - \text{Original Burst Time}$$

---

### 16. What is printed by `cout << "Sequence of processes is:";`?

**Answer:**
It prints the **Gantt Chart sequence**, i.e., the order in which processes are executed over time.

---

### 17. What does the `display()` function do?

**Answer:**
It prints completion, turnaround, and waiting times for all processes, and also computes the **average** of these metrics.

---

### 18. What are the performance metrics calculated in the code?

**Answer:**

- Average Completion Time
- Average Turnaround Time
- Average Waiting Time

---

### 19. Why is Preemptive SJF considered optimal?

**Answer:**
Because it always picks the process that will finish soonest, minimizing **average waiting and turnaround time**.

---

### 20. What is the drawback of SJF scheduling?

**Answer:**
It may cause **starvation** for long processes if short processes keep arriving.

---

### 21. How can starvation in SJF be avoided?

**Answer:**
By using **aging**, which gradually increases the priority of waiting processes.

---

### 22. Is SJF scheduling suitable for real-time systems?

**Answer:**
Not always, because it requires knowing the **burst time** in advance, which is not predictable in real-time systems.

---

### 23. What is the time complexity of this scheduling algorithm in the program?

**Answer:**
For each time unit, we check all processes — so roughly **O(n × total_time)** (inefficient but acceptable for simulation).

---

### 24. What is the difference between Arrival Time and Burst Time?

**Answer:**

- **Arrival Time:** When a process enters the ready queue.
- **Burst Time:** Total CPU time the process requires to complete.

---

### 25. Give a sample output for 3 processes.

**Input:**

```yaml
Enter number of processes: 3
P0: Arrival=0, Burst=5
P1: Arrival=1, Burst=3
P2: Arrival=2, Burst=1
```

**Output:**

```yaml
Sequence of processes: P0 P1 P2 P2 P1 P1 P0 P0 P0

     Completed | Waiting | Turnaround |
P0|  9  |  4  |  9
P1|  5  |  1  |  4
P2|  3  |  0  |  1

Avg completion time: 5.6
Avg turnaround time: 4.6
Avg waiting time: 1.6
```

---

## 🧠 Bonus Conceptual Points (for extra marks!)

| Term | Meaning |
|---|---|
| **Throughput** | Number of processes completed per unit time |
| **Response Time** | Time from arrival to first CPU allocation |
| **Starvation** | Process waits indefinitely due to others |
| **Preemption** | Interrupting a running process for another |
| **Dispatcher** | Component that gives CPU control to selected process |

---

Would you like me to now prepare **another 25 viva questions** that cover **FCFS, Priority, and Round Robin** algorithms too (for your next assignments)?