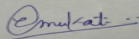
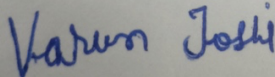
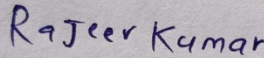




## **Project Title: Intelligent CPU Scheduler Simulator**

### **Student Details:**

<b>Name</b>	<b>Roll No.</b>	<b>Reg. No</b>	<b>Signature</b>
Chandan Mukti	04	12315694	
Varun Joshi	05	12315143	
Rajeev Kumar	06	12317308	

**Supervisor Name: Dr. Gurbinder Singh Brar(31771)**

**LOVELY PROFESSIONAL UNIVERSITY**

**SCHOOL OF COMPUTER SCIENCE**

**Date of Submission : 31/03/2025**

# 1. Project Overview

The "Intelligent CPU Scheduler Simulator" is a tool designed to help us understand how a computer's CPU manages tasks (processes) using different scheduling methods: First-Come-First-Serve (FCFS), Shortest Job First (SJF), Round Robin (RR), and Priority Scheduling. Our goal is to build a simple program where users can input details about processes—like when they arrive, how long they take, and how important they are—and see how each method schedules them. The program will show a text-based Gantt chart (a timeline of when each process runs) and calculate performance metrics like average waiting time (how long processes wait before running) and turnaround time (total time from arrival to completion).

For this version, we're keeping it beginner-friendly by using the console (no fancy graphics yet), but it still meets the core requirements. The expected outcome is a working simulator that's easy to use and educational, showing how these algorithms behave with real inputs. The scope is a single-user, console-based C++ program, with potential to add a GUI later.

## 2. Module-Wise Breakdown

We've split the project into three parts to keep it organized:

### Module 1: User Interface (Console Input/Output)

**Purpose:** This is where we talk to the user—asking for process details and showing results.

**Role:** It's the friendly face of the program, collecting inputs (like how many processes and their details) and displaying the Gantt chart and stats.

### Module 2: Scheduling Logic

**Purpose:** This is the brain that figures out how to schedule processes using the four algorithms.

**Role:** Takes the user's input and runs FCFS, SJF, RR, or Priority to decide the order and timing of processes.

### Module 3: Data Visualization (Text-Based)

**Purpose:** Shows the results in a way we can understand—like a simple timeline and numbers.

**Role:** Turns the scheduling data into a text Gantt chart and calculates waiting/turnaround times.

### 3. Functionalities

Module 1: User Interface (Console Input/Output)\*\*

Feature: Ask for number of processes and their details.

Example:

1."How many processes? 2" then "P1: Arrival = 0, Burst = 5, Priority = 2".

Feature: Let user pick an algorithm.

Example: "Pick: 1. FCFS, 2. SJF, 3. RR, 4. Priority" → User types "3" for Round Robin.

Feature: Show results clearly.

Example: Prints "P1: Waiting = 0, Turnaround = 5" and averages.

Module 2: Scheduling Logic

Feature: FCFS – Runs processes in arrival order.

Example: P1 (arrival 0, burst 5) runs 0-5, P2 (arrival 1, burst 3) runs 5-8.

Feature: SJF – Picks the shortest job ready to go.

Example: P2 (burst 3) runs before P1 (burst 5) if both arrive at 0.

Feature: Round Robin – Gives each process a turn (quantum time).

Example: Quantum = 2, P1 runs 0-2, P2 runs 2-4, P1 runs 4-6, etc.

Feature: Priority – Highest priority goes first.

Example: P1 (priority 2) runs before P2 (priority 1).

Feature: Tracks timing for each process.

Example: Knows P1 starts at 0 and ends at 5 in FCFS.

Module 3: Data Visualization (Text-Based)

Feature: Text Gantt chart.

Example: "| P1 | P2 |" with "0 5 8" below it.

Feature: Calculate and show waiting/turnaround times.

Example: "P1: Waiting = 0, Turnaround = 5; Avg Waiting = 2".

Feature: Simple output format.

Example: Neatly prints each process's stats and averages.

## 4. Technology Used

Programming Languages:

C++: Used for implementing the core scheduling algorithms (FCFS, SJF, Round Robin, and Priority Scheduling). C++ is chosen for its speed and efficiency, making it ideal for beginners to learn and for simulating CPU scheduling logic in a console-based environment.

Python: Employed as the backend language via the Flask framework. Python's simplicity and readability make it perfect for handling server-side logic and integrating the C++ simulator with a web interface.

JavaScript: Powers the frontend interactivity. JavaScript enables dynamic updates to the webpage, such as displaying the Gantt chart and performance metrics in real-time as users interact with the simulator.

HTML: Provides the structure of the web interface. HTML is used to create forms for inputting process details and sections to display results.

CSS (Tailwind CSS): A utility-first CSS framework used for styling the frontend. Tailwind CSS ensures a modern, responsive design with minimal custom CSS, making the interface visually appealing and adaptable to different screen sizes.

### Libraries and Frameworks:

iostream, vector, algorithm (C++ Libraries): Core C++ libraries for input/output, dynamic process storage, and sorting operations in scheduling algorithms.

Flask (Python Framework): A lightweight web framework for Python, used to create a backend server. Flask handles requests from the frontend, runs the C++ scheduler via integration (e.g., through system calls or compiled binaries), and sends results back to the client.

Tailwind CSS: Integrated into HTML for rapid styling. It provides pre-built classes for responsiveness (e.g., mobile-first design) and a clean, professional look without extensive custom CSS.

JavaScript: Used without additional libraries for simplicity, handling DOM manipulation, form submissions, and dynamic rendering of the Gantt chart and metrics.

### Tools and Integration:

GitHub: Used for version control to save and track code changes across the team. The repository (SimpleCPUScheduler) hosts both C++ and web-related files.

Code::Blocks or Visual Studio Code: Editors for writing C++ code (Code::Blocks) and web development (VS Code). VS Code supports Python, HTML, JavaScript, and Tailwind with extensions for a unified development environment.

g++ Compiler: Compiles the C++ code into an executable that the Flask backend can call to perform scheduling simulations.

Python Environment: A virtual environment (e.g., `venv``) to manage Flask and other Python dependencies.

Node.js (Optional): If Tailwind CSS is used via a build process (e.g., with a CDN or local setup), Node.js might be used to manage frontend dependencies, though a CDN approach keeps it simple for this project.

#### Integration Approach:

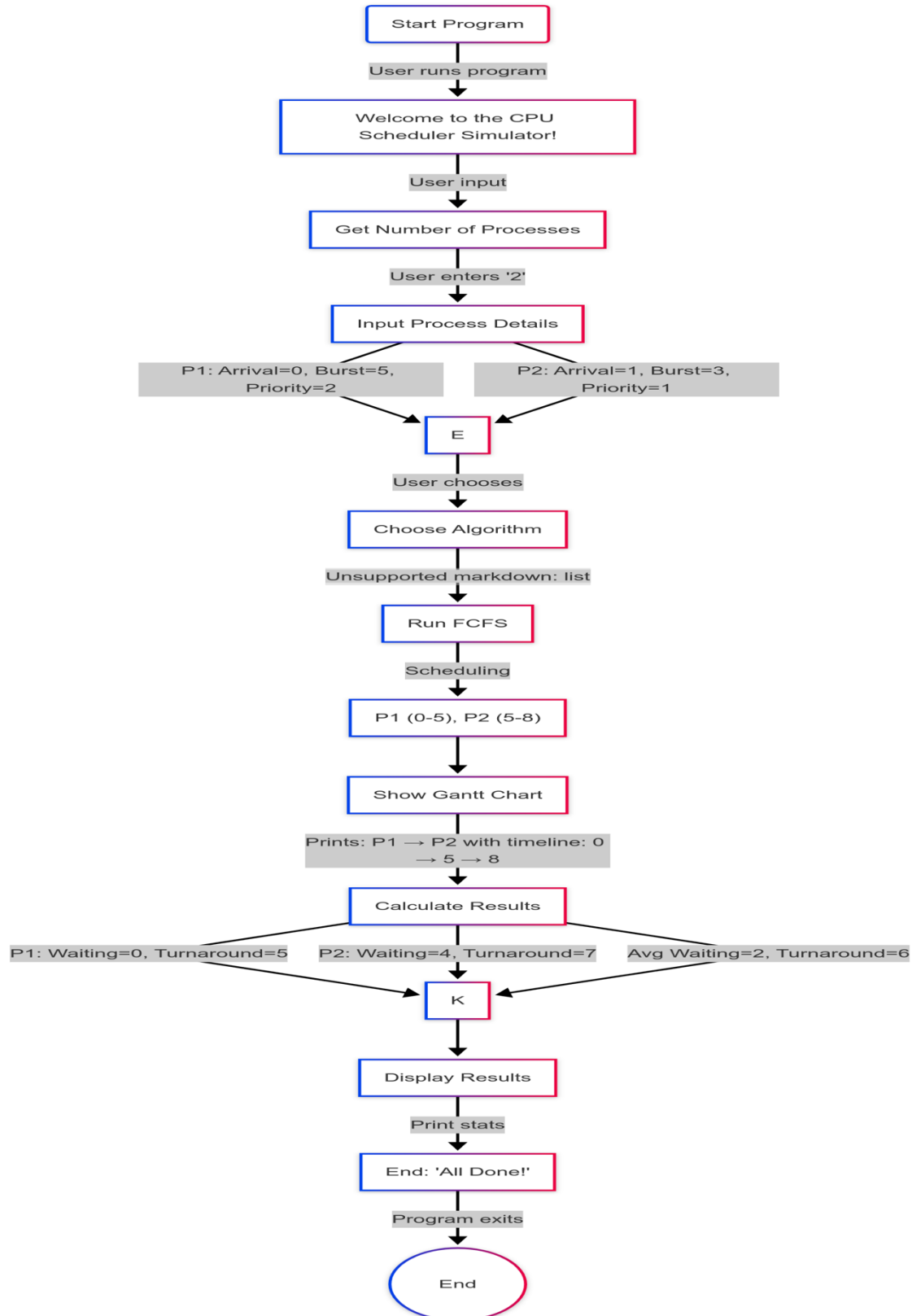
The C++ simulator is compiled into an executable (e.g., `scheduler.exe``). The Flask backend runs this executable using system calls (e.g., `subprocess`` in Python), passing user inputs as arguments and capturing the output (Gantt chart and metrics).

Flask serves a web API (e.g., `/simulate``) that the JavaScript frontend calls via `fetch`` or `XMLHttpRequest``, sending process details and receiving scheduling results as JSON.

The frontend uses HTML for layout, Tailwind CSS for responsive styling, and JavaScript to dynamically render the Gantt chart and metrics in the browser.

## 5. Flow Diagram

Here's how the program works step-by-step, based on a scenario where a user runs it with 2 processes and picks FCFS:



## 6. Revision Tracking on GitHub

Repository Name: SimpleCPUScheduler

GitHub Link: <https://github.com/varunjoshi84/SimpleCPUScheduler>

Tracking Plan:

Commit 1: Basic setup -empty main() and Process struct.

Commit 2: Add user input code (asking for processes).

Commit 3: Add FCFS with Gantt and results.

Commit 4: Add SJF, RR, and Priority one by one.

Commit 5: Final version with all algorithms tested.

## 7. Conclusion and Future Scope

This simulator is a cool way to see how CPU scheduling works without needing fancy tools. It's simple, runs in the console, and shows everything we need—Gantt charts and performance numbers. We learned how each algorithm handles processes differently, like FCFS being fair but slow, or SJF being quick but picky. For the future, we could:

Add a graphical window (using Qt or SFML) to make it prettier.

Let it handle processes that can be interrupted (pre-emptive versions).

Save results to a file for later.

Add more algorithms like Multi-Level Queue.

## 8. References

Books:

Operating System Concepts\* by Silber Schatz (for scheduling ideas).

C++ Primer by Lippman (to learn C++ basics).

Websites:

C++ Reference: <https://en.cppreference.com/w/>

## Appendix

### A.AI-Generated Project Elaboration/Breakdown Report

#### 1. Introduction

The Intelligent CPU Scheduler Simulator is designed to provide an interactive way to understand and analyze different CPU scheduling algorithms. This project aims to help users visualize scheduling operations and performance metrics such as waiting time, turnaround time, and response time.

#### 2. Problem Statement

In operating systems, CPU scheduling plays a crucial role in ensuring efficient utilization of processor time. Different scheduling algorithms impact system performance in various ways. The challenge is to build a simulator that allows users to input process details and visualize how different scheduling techniques execute these processes dynamically.

#### 3. Objectives

The key objectives of this project are:

1. To implement four primary CPU scheduling algorithms:
  - First-Come-First-Serve (FCFS) – Simple, non-preemptive scheduling.
  - Shortest Job First (SJF) – Optimized for minimal waiting time.
  - Round Robin (RR) – Fair scheduling for time-sharing systems.
  - Priority Scheduling – Ensures high-priority processes execute first.
2. To simulate process execution and display scheduling results using a text-based Gantt chart.
3. To compute and analyze performance metrics (waiting time, turnaround time).
4. To provide a modular and extendable design for adding future enhancements like preemptive scheduling.

#### 4. System Architecture

The system follows a modular structure:

- Input Module: Accepts process details (arrival time, burst time, priority).
- Processing Module: Implements the chosen scheduling algorithm.



- Output Module: Displays execution order, Gantt chart, and calculated metrics.

## 5. Functional Breakdown

### Functionality Description

User Input	Allows users to enter process details.
Algorithm Selection	Users can choose from FCFS, SJF, RR, or Priority.
Scheduling Execution	Implements and executes the selected algorithm.
Gantt Chart Generation	Displays process execution order visually.
Performance Calculation	Computes waiting and turnaround times.

## 6. Implementation Details

- Programming Language: C++
- Libraries Used: iostream, vector, algorithm
- Development Tools: GitHub (version control), Code::Blocks/VS Code (IDE), g++ compiler

## 7. Expected Outcome

- A fully functional CPU scheduling simulator.
- Accurate scheduling execution with visual Gantt chart representation.
- Calculation of key scheduling performance metrics.
- Improved understanding of CPU scheduling techniques for students and researchers.

## B. Problem Statement

"Intelligent CPU Scheduler Simulator "

Description: Develop a simulator for CPU scheduling algorithms (FCFS, SJF, Round Robin, Priority Scheduling) with real-time visualizations. The simulator should allow users to input processes with arrival times, burst times, and priorities and visualize Gantt charts and performance metrics like average waiting time and turnaround time."

## C. Solution/Code

### Front-End code(Html code for front end Structure)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CPU Scheduler Simulator</title>
  <script src="https://cdn.tailwindcss.com"></script>
  <style>
    .gantt-bar {
      height: 40px;
      margin: 8px 0;
      color: white;
      text-align: center;
      line-height: 40px;
      display: inline-block;
      transition: width 0.5s ease-in-out;
      position: relative;
    }
    .gantt-bar:hover::after {
      content: attr(data-tooltip);
      position: absolute;
      bottom: 45px;
      left: 50%;
      transform: translateX(-50%);
      background: rgba(0, 0, 0, 0.8);
      color: white;
      padding: 4px 8px;
      border-radius: 4px;
      font-size: 12px;
      white-space: nowrap;
    }
    .input-error { border-color: #ef4444; }
    .hidden { display: none; }
    .fade-in { animation: fadeIn 0.5s; }
    @keyframes fadeIn { from { opacity: 0; } to { opacity: 1; } }
  </style>
</head>
<body class="bg-gray-100 font-sans min-h-screen flex items-center justify-center">
  <div class="container mx-auto p-6 max-w-4xl">
    <h1 class="text-4xl font-bold text-center text-blue-600 mb-8 animate-bounce">CPU Scheduler Simulator</h1>
```

```

<!-- Form Section -->
<div class="bg-white p-6 rounded-lg shadow-lg">
  <h2 class="text-2xl font-semibold mb-6 text-gray-800">Enter Process
Details</h2>
  <div class="mb-6">
    <label class="block text-gray-700 font-medium">Number of
Processes:</label>
    <input type="number" id="numProcesses" min="1" max="10" class="w-
full p-3 border rounded mt-2 focus:ring-2 focus:ring-blue-500" placeholder="e.g.,
4">
  </div>
  <div id="processInputs" class="space-y-6"></div>
  <div class="mb-6">
    <label class="block text-gray-700 font-medium">Scheduling
Algorithm:</label>
    <select id="algorithm" class="w-full p-3 border rounded mt-2
focus:ring-2 focus:ring-blue-500">
      <option value="1">FCFS</option>
      <option value="2">SJF</option>
      <option value="3">Round Robin</option>
      <option value="4">Priority</option>
    </select>
  </div>
  <div id="quantumDiv" class="mb-6 hidden">
    <label class="block text-gray-700 font-medium">Time Quantum (for
Round Robin):</label>
    <input type="number" id="quantum" min="1" class="w-full p-3 border
rounded mt-2 focus:ring-2 focus:ring-blue-500" placeholder="e.g., 2">
  </div>
  <div class="flex space-x-4">
    <button id="submitBtn" class="bg-blue-500 text-white p-3 rounded-lg
hover:bg-blue-600 transition-colors flex-1">Run Simulation</button>
    <button id="resetBtn" class="bg-gray-500 text-white p-3 rounded-lg
hover:bg-gray-600 transition-colors flex-1">Reset</button>
  </div>
  <p id="loading" class="text-gray-500 mt-4 hidden">Running
simulation...</p>
</div>

<!-- Results Section -->
<div id="results" class="mt-8 bg-white p-6 rounded-lg shadow-lg hidden
fade-in">
  <h2 class="text-2xl font-semibold mb-6 text-gray-800">Simulation
Results</h2>
  <div class="flex justify-center space-x-4 mb-6">
    <button id="showGantt" class="bg-blue-500 text-white p-2 rounded
hover:bg-blue-600">Show Gantt Chart</button>
    <button id="showMetrics" class="bg-blue-500 text-white p-2 rounded
hover:bg-blue-600">Show Metrics</button>
  </div>
  <div id="ganttChart" class="mb-6"></div>

```

```

        <div id="metrics" class="text-gray-700 hidden"></div>
    </div>
</div>

<script src="/static/script.js"></script>
</body>
</html>

```

Script Code(Javascript code for dom manipulation and responsiveness)

```

// Toggle quantum visibility
document.getElementById('algorithm').addEventListener('change', function() {
    const algo = this.value;
    document.getElementById('quantumDiv').classList.toggle('hidden', algo !== '3');
    updateProcessInputs();
});

// Generate process input fields dynamically
document.getElementById('numProcesses').addEventListener('input',
updateProcessInputs);

function updateProcessInputs() {
    const num = parseInt(document.getElementById('numProcesses').value) || 0;
    const algo = document.getElementById('algorithm').value;
    const showPriority = algo === '4';
    const container = document.getElementById('processInputs');
    container.innerHTML = '';

    for (let i = 0; i < num; i++) {
        let inputs = `
            <div class="grid grid-cols-1 md:grid-cols-${showPriority ? '3' : '2'}
gap-4">
                <div>
                    <label class="block text-gray-700">P${i + 1} Arrival
Time:</label>
                    <input type="number" id="arrival${i}" min="0" class="w-full p-3
border rounded mt-2" placeholder="e.g., 0" required>
                </div>
                <div>
                    <label class="block text-gray-700">P${i + 1} Burst
Time:</label>
                    <input type="number" id="burst${i}" min="1" class="w-full p-3
border rounded mt-2" placeholder="e.g., 5" required>
                </div>
            `;
        if (showPriority) {
            inputs += `
                <div>
                    <label class="block text-gray-700">P${i + 1} Priority:</label>

```

```

        <input type="number" id="priority${i}" min="0" class="w-full p-
3 border rounded mt-2" placeholder="e.g., 2" required>
        </div>
    `;
    }
    inputs += `</div>`;
    container.innerHTML += inputs;
}

// Add real-time validation
document.querySelectorAll('input[type="number"]').forEach(input => {
    input.addEventListener('input', function() {
        if (!this.value || parseInt(this.value) < parseInt(this.min)) {
            this.classList.add('input-error');
        } else {
            this.classList.remove('input-error');
        }
    });
});
});
}

// Handle form submission
document.getElementById('submitBtn').addEventListener('click', async function() {
    const num = parseInt(document.getElementById('numProcesses').value);
    const algo = document.getElementById('algorithm').value;
    const quantum = algo === '3' ? document.getElementById('quantum').value : 0;

    let inputData = `${num}\n${algo}\n${quantum}\n`;
    for (let i = 0; i < num; i++) {
        const arrival = document.getElementById(`arrival${i}`).value;
        const burst = document.getElementById(`burst${i}`).value;
        const priority = algo === '4' ?
document.getElementById(`priority${i}`).value : '0';
        if (!arrival || !burst || (algo === '4' && !priority)) {
            alert(`Please fill all fields for Process P${i + 1}`);
            return;
        }
        inputData += `${arrival} ${burst} ${priority}\n`;
    }

    document.getElementById('loading').classList.remove('hidden');
    try {
        const output = await runSimulation(inputData);
        displayResults(output);
    } catch (error) {
        alert("Error running simulation: " + error.message);
    } finally {
        document.getElementById('loading').classList.add('hidden');
    }
});
});

```

```

// Run simulation by sending input to Flask server
async function runSimulation(inputData) {
  const response = await fetch('http://127.0.0.1:5001/run', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ input: inputData })
  });
  if (!response.ok) throw new Error('Simulation failed');
  const data = await response.json();
  return data.output;
}

// Display results with dynamic Gantt chart
function displayResults(output) {
  const lines = output.trim().split('\n');
  const ganttChart = document.getElementById('ganttChart');
  const metrics = document.getElementById('metrics');
  ganttChart.innerHTML = '';
  metrics.innerHTML = '';

  const colors = ['bg-blue-500', 'bg-green-500', 'bg-purple-500', 'bg-red-500',
    'bg-yellow-500', 'bg-teal-500'];
  if (lines[0].startsWith('|')) {
    const processes = lines[0].split('|').filter(p => p.trim());
    const times = lines[1].split(/\s+/).filter(t => t.trim());
    for (let i = 0; i < processes.length; i++) {
      const start = parseInt(times[i]);
      const end = parseInt(times[i + 1]);
      const width = (end - start) * 20;
      const color = colors[i % colors.length];
      ganttChart.innerHTML += `
        <div class="gantt-bar ${color}" style="width: ${width}px" data-
        tooltip="P${processes[i].trim().split(' ')[1]}: ${start}-
        ${end}">${processes[i]}</div>
      `;
    }
  }

  metrics.innerHTML = lines.slice(2).map(line => `<p>${line}</p>`).join('');
  document.getElementById('results').classList.remove('hidden');
  document.getElementById('ganttChart').classList.remove('hidden');
  document.getElementById('metrics').classList.add('hidden');
}

// Toggle between Gantt and Metrics
document.getElementById('showGantt').addEventListener('click', function() {
  document.getElementById('ganttChart').classList.remove('hidden');
  document.getElementById('metrics').classList.add('hidden');
});

document.getElementById('showMetrics').addEventListener('click', function() {

```

```

    document.getElementById('ganttChart').classList.add('hidden');
    document.getElementById('metrics').classList.remove('hidden');
});

// Reset form
document.getElementById('resetBtn').addEventListener('click', function() {
    document.getElementById('numProcesses').value = '';
    document.getElementById('algorithm').value = '1';
    document.getElementById('quantum').value = '';
    document.getElementById('processInputs').innerHTML = '';
    document.getElementById('results').classList.add('hidden');
    document.getElementById('quantumDiv').classList.add('hidden');
});

```

Python code(Connect frontend to backend)

```

from flask import Flask, render_template, jsonify, request
import subprocess
import os

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/run', methods=['POST'])
def run_simulation():
    data = request.json
    if not data or 'input' not in data:
        return jsonify({"error": "Missing input data"}), 400

    # Write input data to input.txt
    with open('input.txt', 'w') as f:
        f.write(data['input'])

    # Run the C++ scheduler
    try:
        subprocess.run(['./scheduler'], check=True)
    except subprocess.CalledProcessError as e:
        return jsonify({"error": "Scheduler execution failed", "details": str(e)}),
500

    # Read the output from output.txt
    if os.path.exists('output.txt'):
        with open('output.txt', 'r') as f:
            output = f.read()
        return jsonify({"output": output})
    else:

```

```

        return jsonify({"error": "Output file not generated"}), 500

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5001)

```

## Scheduling Algorithm(CPP)

```

#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
using namespace std;

struct Process {
    int pid, arrival, burst, priority, remaining;
    Process(int p, int a, int b, int pr, int r) : pid(p), arrival(a), burst(b),
priority(pr), remaining(r) {}
};

struct GanttPiece {
    int pid, start, end;
    GanttPiece(int p, int s, int e) : pid(p), start(s), end(e) {}
};

void showGantt(vector<GanttPiece>& gantt, ofstream& out) {
    out << "|";
    for (int i = 0; i < gantt.size(); i++) {
        out << " P" << gantt[i].pid << " |";
    }
    out << "\n0";
    for (int i = 0; i < gantt.size(); i++) {
        out << "    " << gantt[i].end;
    }
    out << "\n";
}

void calculateResults(vector<Process>& processes, vector<GanttPiece>& gantt,
ofstream& out) {
    float totalWaiting = 0, totalTurnaround = 0;
    int n = processes.size();
    for (int i = 0; i < n; i++) {
        int finishTime = 0;
        for (int j = 0; j < gantt.size(); j++) {
            if (gantt[j].pid == processes[i].pid && gantt[j].end > finishTime) {
                finishTime = gantt[j].end;
            }
        }
        int turnaround = finishTime - processes[i].arrival;
        int waiting = turnaround - processes[i].burst;
        out << "P" << processes[i].pid << ": Waiting Time = " << waiting

```



```

        << ", Turnaround Time = " << turnaround << "\n";
        totalWaiting += waiting;
        totalTurnaround += turnaround;
    }
    out << "Average Waiting Time = " << totalWaiting / n << "\n";
    out << "Average Turnaround Time = " << totalTurnaround / n << "\n";
}

void doFCFS(vector<Process>& processes, ofstream& out) {
    vector<GanttPiece> gantt;
    int currentTime = 0;
    for (int i = 0; i < processes.size(); i++) {
        if (currentTime < processes[i].arrival) currentTime = processes[i].arrival;
        GanttPiece piece(processes[i].pid, currentTime, currentTime +
processes[i].burst);
        gantt.push_back(piece);
        currentTime += processes[i].burst;
    }
    showGantt(gantt, out);
    calculateResults(processes, gantt, out);
}

void doSJF(vector<Process>& processes, ofstream& out) {
    vector<GanttPiece> gantt;
    int currentTime = 0, done = 0;
    vector<bool> finished(processes.size(), false);
    while (done < processes.size()) {
        int shortest = -1;
        int minBurst = 9999;
        for (int i = 0; i < processes.size(); i++) {
            if (!finished[i] && processes[i].arrival <= currentTime &&
processes[i].burst < minBurst) {
                shortest = i;
                minBurst = processes[i].burst;
            }
        }
        if (shortest == -1) {
            currentTime++;
        } else {
            GanttPiece piece(processes[shortest].pid, currentTime, currentTime +
processes[shortest].burst);
            gantt.push_back(piece);
            currentTime += processes[shortest].burst;
            finished[shortest] = true;
            done++;
        }
    }
    showGantt(gantt, out);
    calculateResults(processes, gantt, out);
}

```

```

void doRoundRobin(vector<Process>& processes, int quantum, ofstream& out) {
    vector<GanttPiece> gantt;
    int currentTime = 0, done = 0;
    vector<Process> proc = processes;
    while (done < processes.size()) {
        bool didSomething = false;
        for (int i = 0; i < proc.size(); i++) {
            if (proc[i].remaining > 0 && proc[i].arrival <= currentTime) {
                didSomething = true;
                int runTime = min(quantum, proc[i].remaining);
                GanttPiece piece(proc[i].pid, currentTime, currentTime + runTime);
                gantt.push_back(piece);
                currentTime += runTime;
                proc[i].remaining -= runTime;
                if (proc[i].remaining == 0) done++;
            }
        }
        if (!didSomething) currentTime++;
    }
    showGantt(gantt, out);
    calculateResults(processes, gantt, out);
}

void doPriority(vector<Process>& processes, ofstream& out) {
    vector<GanttPiece> gantt;
    int currentTime = 0, done = 0;
    vector<bool> finished(processes.size(), false);
    while (done < processes.size()) {
        int best = -1;
        int maxPriority = -1;
        for (int i = 0; i < processes.size(); i++) {
            if (!finished[i] && processes[i].arrival <= currentTime &&
processes[i].priority > maxPriority) {
                best = i;
                maxPriority = processes[i].priority;
            }
        }
        if (best == -1) {
            currentTime++;
        } else {
            GanttPiece piece(processes[best].pid, currentTime, currentTime +
processes[best].burst);
            gantt.push_back(piece);
            currentTime += processes[best].burst;
            finished[best] = true;
            done++;
        }
    }
    showGantt(gantt, out);
    calculateResults(processes, gantt, out);
}

```

```

int main() {
    ifstream inFile("input.txt");
    ofstream outFile("output.txt");
    if (!inFile || !outFile) {
        cerr << "Error opening files!\n";
        return 1;
    }

    int numProcesses, algo, quantum;
    inFile >> numProcesses >> algo >> quantum;
    vector<Process> processes;
    for (int i = 0; i < numProcesses; i++) {
        int arrival, burst, priority;
        inFile >> arrival >> burst >> priority;
        Process p(i + 1, arrival, burst, priority, burst);
        processes.push_back(p);
    }

    switch (algo) {
        case 1: doFCFS(processes, outFile); break;
        case 2: doSJF(processes, outFile); break;
        case 3: doRoundRobin(processes, quantum, outFile); break;
        case 4: doPriority(processes, outFile); break;
        default: outFile << "Invalid algorithm choice!\n";
    }

    inFile.close();
    outFile.close();
    return 0;
}

```

**OUTPUT:**

## CPU Scheduler Simulator

### Enter Process Details

Number of Processes:

e.g., 4

Scheduling Algorithm:

FCFS

Run Simulation

Reset

## CPU Scheduler Simulator

### Enter Process Details

Number of Processes:

4

P1 Arrival Time:

0

P1 Burst Time:

6

P2 Arrival Time:

0

P2 Burst Time:

7

P3 Arrival Time:

2

P3 Burst Time:

9

P4 Arrival Time:

4

P4 Burst Time:

10

Scheduling Algorithm:

FCFS

Run Simulation

Reset

### Simulation Results

Show Gantt Chart

Show Metrics

P1

P2

P3

P4

127.0.0.1:5001

CPU Scheduler Simulator

### Enter Process Details

Number of Processes: 4

P1 Arrival Time: 0	P1 Burst Time: 6
P2 Arrival Time: 0	P2 Burst Time: 7
P3 Arrival Time: 2	P3 Burst Time: 9
P4 Arrival Time: 4	P4 Burst Time: 10

Scheduling Algorithm: FCFS

Run Simulation Reset

### Simulation Results

Show Gantt Chart Show Metrics

P1: Waiting Time = 0, Turnaround Time = 6  
P2: Waiting Time = 6, Turnaround Time = 13  
P3: Waiting Time = 11, Turnaround Time = 20  
P4: Waiting Time = 18, Turnaround Time = 28  
Average Waiting Time = 8.75  
Average Turnaround Time = 16.75

esproject

main.cpp index.html JS script.js .gitignore requirements.txt M haha.py 2, U app.py 1 scheduler.cpp input.txt M

EXPLORER

- main.cpp
- index.html
- templates
- script.js
- static
- requirements.txt
- haha.py
- app.py
- scheduler.cpp
- input.txt

OUTLINE

TIMELINE

OSPROJECT

- venv
- static
- script.js
- templates
- index.html
- requirements.txt
- app.py
- haha.py
- input.txt
- main.cpp
- output.txt
- project.txt
- README.md
- requirements.txt
- scheduler
- tempCodeRunnerFile.cpp

```
113 void doPriority(vector<Process*> processes, ofstream& out) {
114     while (true) {
115         // ...
116     }
117     showGantt(gantt, out);
118     calculateResults(processes, gantt, out);
119 }
120
121 int main() {
122     ifstream inFile("input.txt");
123     ofstream outFile("output.txt");
124     if (!inFile || !outFile) {
125         cerr << "Error opening files!\n";
126         return 1;
127     }
128
129     int numProcesses, algo, quantum;
130     inFile >> numProcesses >> algo >> quantum;
131     vector<Process*> processes;
132     for (int i = 0; i < numProcesses; i++) {
133         int arrival, burst, priority;
134         inFile >> arrival >> burst >> priority;
135         Process p{i, arrival, burst, priority, burst};
136         processes.push_back(p);
137     }
138
139     switch (algo) {
140         case 1: doFCFS(processes, outFile); break;
141         case 2: doSJF(processes, outFile); break;
142         case 3: doRoundRobin(processes, quantum, outFile); break;
143         case 4: doPriority(processes, outFile); break;
144         default: outFile << "Invalid algorithm choice!\n";
145     }
146 }
147
148 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
```

Python + Python 3.10.10

WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.

- Running on all addresses (0.0.0.0)
- Running on http://127.0.0.1:5001
- Running on http://172.22.121.148:5001

Press CTRL-C to quit

- Serving Flask app 'app'
- Debug mode: on
- Restarting with stat
- Debugger is active!
- Debugger PID: 121-893-546

```
127.0.0.1 - - [30/Mar/2025 20:52:58] "GET /apple-touch-icon-precomposed.png HTTP/1.1" 404 -
127.0.0.1 - - [30/Mar/2025 20:52:58] "GET /apple-touch-icon.png HTTP/1.1" 404 -
127.0.0.1 - - [30/Mar/2025 20:52:58] "GET /favicon.ico HTTP/1.1" 200 -
127.0.0.1 - - [30/Mar/2025 20:53:00] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [30/Mar/2025 20:53:00] "GET /static/scripts.js HTTP/1.1" 304 -
127.0.0.1 - - [30/Mar/2025 20:53:45] "POST /run HTTP/1.1" 200 -
```