Project Operating System..

Prepared by:

Shams Amr Saber

Project Introduction

This project aims to **simulate CPU scheduling algorithms** using **Python**, with a graphical user interface (GUI) developed using the **Tkinter** library, and **Matplotlib** for visualizing the Gantt chart.
The simulation allows users to experiment with and analyze the performance of four popular scheduling algorithms:

FCFS (First-Come, First-Served)

SJF (Shortest Job First)

Priority Scheduling

Round Robin

The project generates random processes at the click of the "Generate Processes" button, with processes automatically named (e.g., P1, P2, P3, etc.). After selecting a scheduling algorithm, the user can press the "Run Algorithm" button to execute it, displaying the results visually in the Gantt chart and in tables showing details for each process, including:

Start Time

Finish Time

Waiting Time

Turnaround Time

The project also calculates the average waiting and turnaround times, enabling a direct comparison of the performance of different scheduling algorithms.

The goal of this project is to **understand the behavior of different scheduling algorithms** and provide students and developers with an intuitive, visual way to learn and compare these algorithms.

1. FCFS (First-Come, First-Served)

Description:

FCFS is one of the simplest scheduling algorithms. In FCFS, processes are executed in the order they arrive in the ready queue. The first process to arrive is the first to be executed.

How it works:

Order of Execution: Processes are executed in the order they arrive.

Non-preemptive: Once a process starts

executing, it runs to completion without interruption.

Process Completion: A process will execute until

it finishes its burst time (execution time).

Waiting Time: The waiting time for each process is calculated by the time a process has to wait before it begins execution.

Turnaround Time: This is the total time taken for a process to complete from the moment it enters the queue to its completion time.

Calculations:

Start Time: The start time of each process is

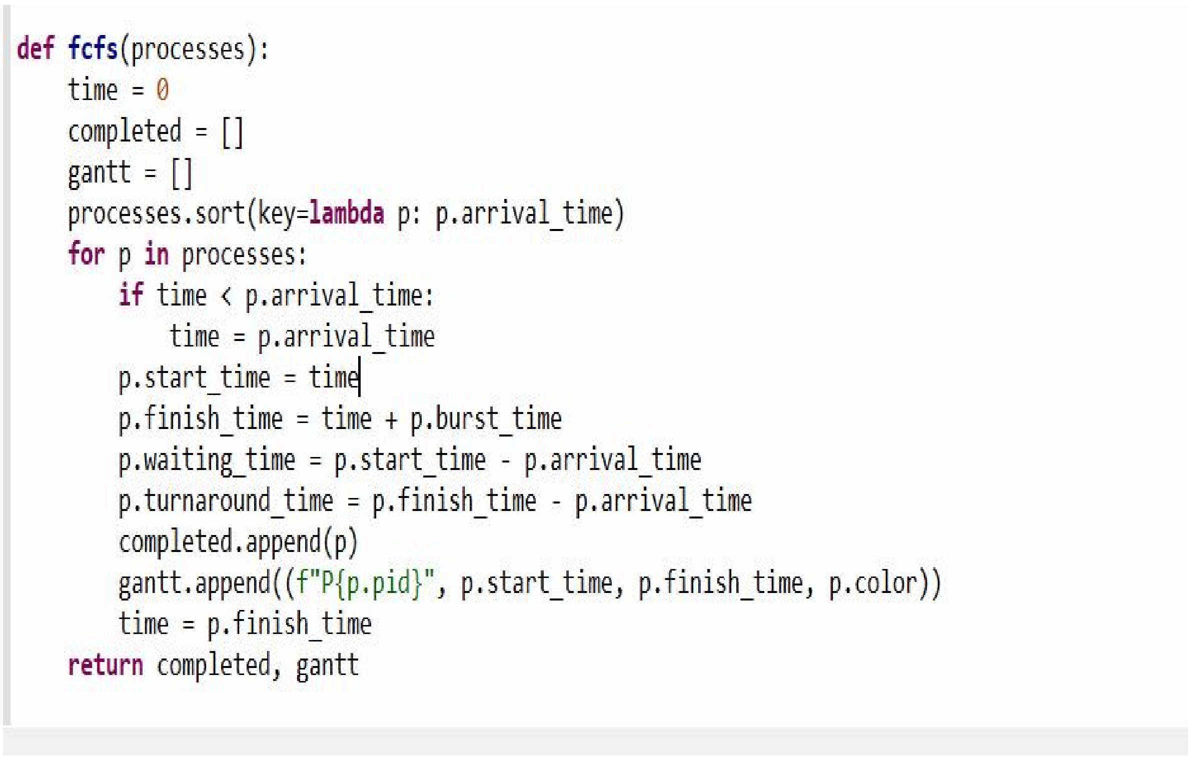
calculated based on the finish time of the previous process.

Finish Time: The finish time is the start time plus the

burst time of the process.

Waiting Time: Waiting time is the difference between the start time and the arrival time.

Turnaround Time: The turnaround time is the difference between the finish time and the arrival time.



```python
def fcfs(processes):
    time = 0
    completed = []
    gantt = []
    processes.sort(key=lambda p: p.arrival_time)
    for p in processes:
        if time < p.arrival_time:
            time = p.arrival_time
        p.start_time = time
        p.finish_time = time + p.burst_time
        p.waiting_time = p.start_time - p.arrival_time
        p.turnaround_time = p.finish_time - p.arrival_time
        completed.append(p)
        gantt.append((f"P{p.pid}", p.start_time, p.finish_time, p.color))
        time = p.finish_time
    return completed, gantt
```

2. SJF (Shortest Job First)

Description:

SJF scheduling algorithm executes the process with the smallest execution time first. This method reduces the average waiting time in the system, but it requires knowing or estimating the burst time of each process, which may not always be possible.

How it works:

Non-preemptive or Preemptive: SJF can be either non-preemptive or preemptive. In the **non-preemptive** version, once a process starts executing, it runs to completion. In the **preemptive** version, if a new process arrives with a shorter burst time, the current process is preempted and the new one is scheduled.

Execution Order: The process with the smallest burst time is executed first. If two processes have the same burst time, they are scheduled in the order they arrive.

Potential Issues: One problem with SJF is that short jobs get processed faster, but long jobs might face starvation if there are always shorter jobs arriving.

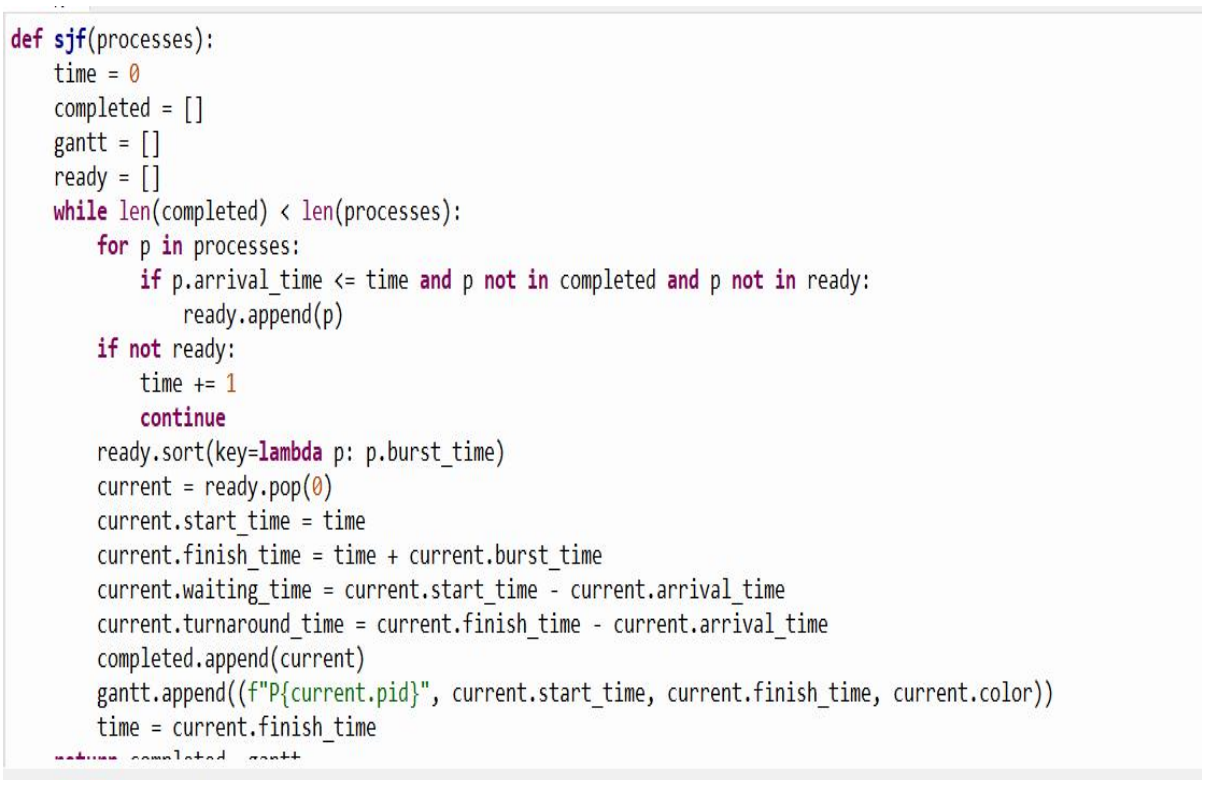
Calculations:

Start Time: The start time of each process is based on the previous process's finish time and the length of the burst times.

Finish Time: The finish time is calculated as the start time plus the burst time.

Waiting Time: The waiting time is calculated as the difference between the finish time and the arrival time minus the burst time.

Turnaround Time: The turnaround time is the difference between the finish time and the arrival time.



```python
def sjf(processes):
    time = 0
    completed = []
    gantt = []
    ready = []
    while len(completed) < len(processes):
        for p in processes:
            if p.arrival_time <= time and p not in completed and p not in ready:
                ready.append(p)
        if not ready:
            time += 1
            continue
        ready.sort(key=lambda p: p.burst_time)
        current = ready.pop(0)
        current.start_time = time
        current.finish_time = time + current.burst_time
        current.waiting_time = current.start_time - current.arrival_time
        current.turnaround_time = current.finish_time - current.arrival_time
        completed.append(current)
        gantt.append((f"P{current.pid}", current.start_time, current.finish_time, current.color))
        time = current.finish_time
    return completed, gantt
```

3. Priority Scheduling

Description:

Priority Scheduling assigns a priority to each process, and processes are executed based on their priority. The process with the highest priority is executed first. The priority can be assigned based on various factors such as resource requirements, time limits, or external conditions.

How it works:

Preemptive or Non-preemptive: Priority scheduling can be either preemptive or non-preemptive. In the **preemptive** version, a process can be interrupted if a higher priority process arrives. In the **non-preemptive** version, once a process starts execution, it runs to completion.

Execution Order: Processes are scheduled according to their priority, where the process with the highest priority (lowest numerical value for priority) is executed first.

Starvation: A potential issue with this algorithm is that low-priority processes may never get executed if high-priority processes continuously arrive.

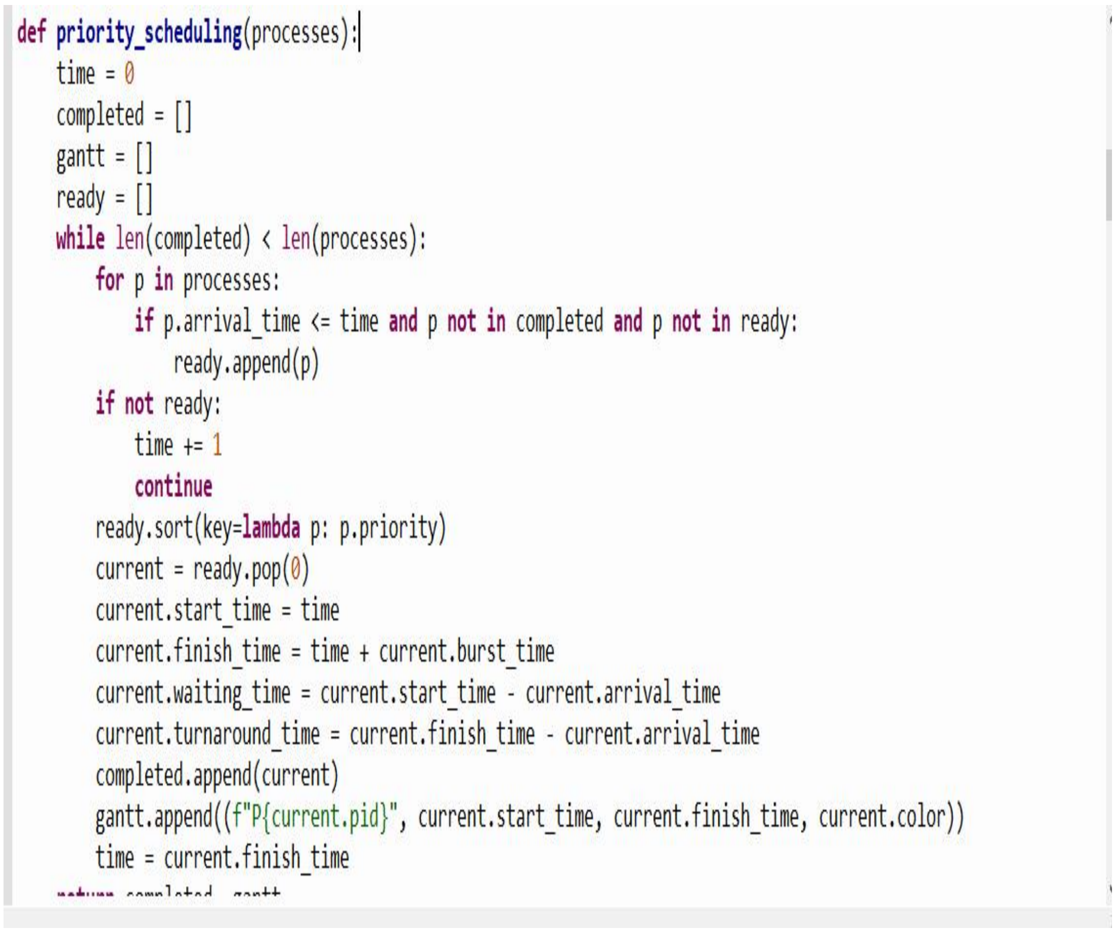
Calculations:

Start Time: The start time is based on the previous process's completion time, adjusted by the priority **scheduling.**

Finish Time: The finish time is calculated as the start time plus the burst time of the process.

Waiting Time: Waiting time is the time the process spends waiting in the queue before execution starts. It is calculated by subtracting the arrival time and burst time from the finish time.

Turnaround Time: The turnaround time is the difference between the finish time and the arrival time.



```python
def priority_scheduling(processes):
    time = 0
    completed = []
    gantt = []
    ready = []
    while len(completed) < len(processes):
        for p in processes:
            if p.arrival_time <= time and p not in completed and p not in ready:
                ready.append(p)
        if not ready:
            time += 1
            continue
        ready.sort(key=lambda p: p.priority)
        current = ready.pop(0)
        current.start_time = time
        current.finish_time = time + current.burst_time
        current.waiting_time = current.start_time - current.arrival_time
        current.turnaround_time = current.finish_time - current.arrival_time
        completed.append(current)
        gantt.append((f"P{current.pid}", current.start_time, current.finish_time, current.color))
        time = current.finish_time
    return completed, gantt
```

4. Round Robin (RR)

Description:

Round Robin is one of the most commonly used CPU scheduling algorithms. It allocates a fixed time quantum (time slice) for each process, and each process is executed for this time period. If the process does not finish execution in the given time quantum, it is preempted, and the CPU moves to the next process in the ready queue.

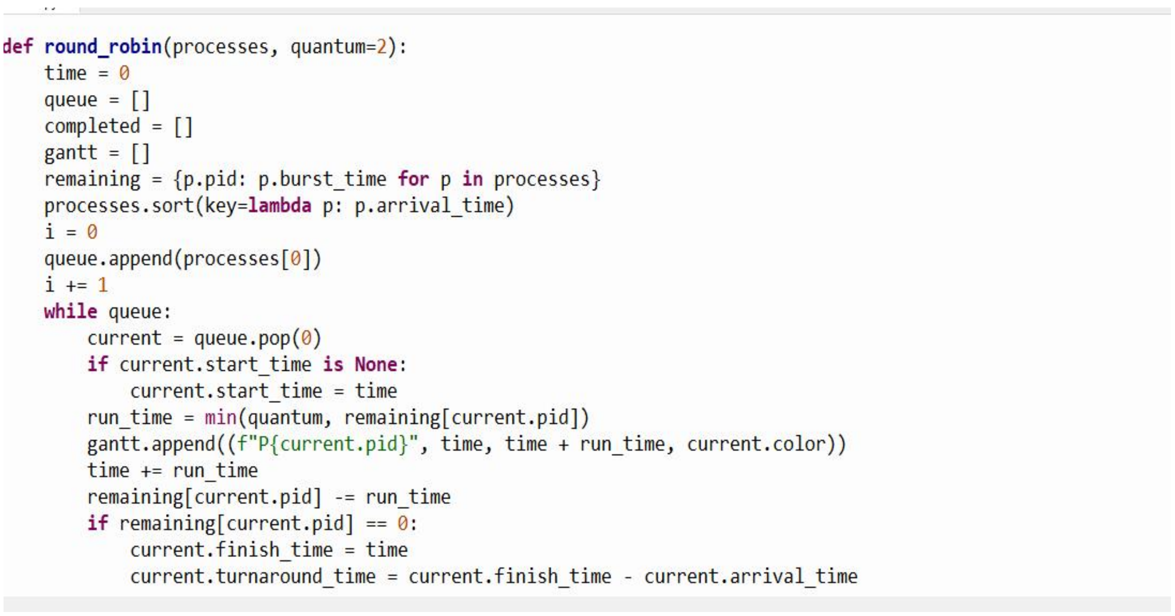
How it works:

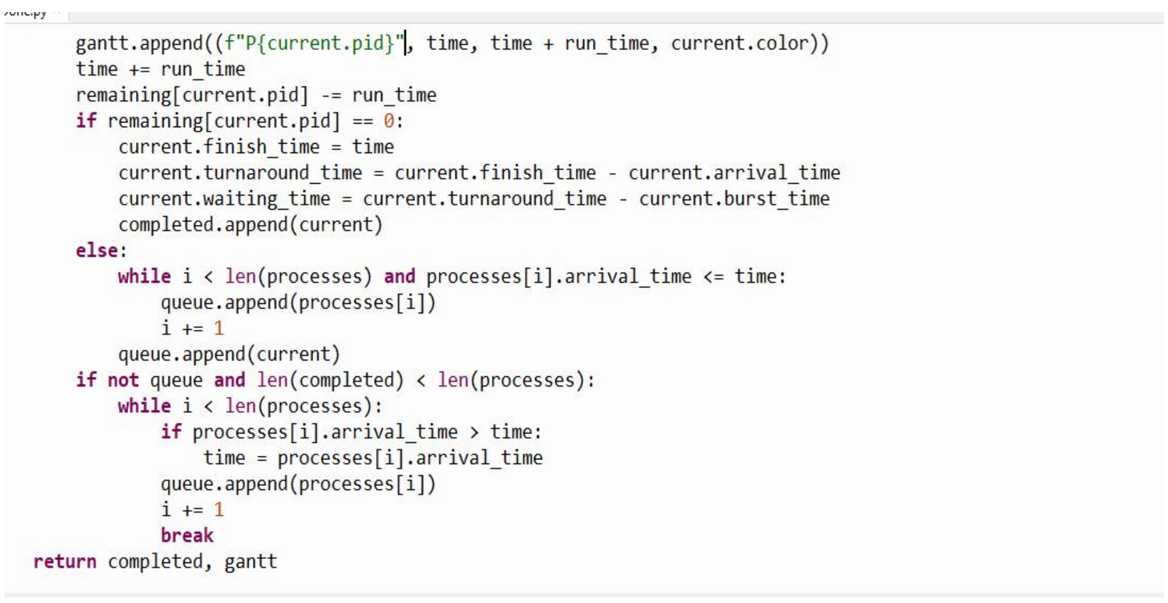
- **Time Quantum:** Each process gets an equal time slice (time quantum). After this time period, if the process is not completed, it is moved to the back of the ready queue.
- **Preemptive:** Round Robin is a preemptive algorithm, meaning that processes can be interrupted and resumed later.
- **Fair Scheduling**: This algorithm is considered fair because each process gets an equal amount of CPU time.
- **Completion**: Once a process completes its execution, it is removed from the queue.

Calculations:

- **Start Time:** The start time of each process is determined by the order in which it is scheduled.
- **Finish Time**: The finish time is calculated as the total time the process has been executing, considering multiple time quanta, if necessary.
- **Waiting Time**: Waiting time is the total time a process waits in the ready queue before its turn for execution.

- **Turnaround Time:** Turnaround time is the total time from the arrival of the process to its completion, including time spent waiting and executing.


```python
def round_robin(processes, quantum=2):
    time = 0
    queue = []
    completed = []
    gantt = []
    remaining = {p.pid: p.burst_time for p in processes}
    processes.sort(key=lambda p: p.arrival_time)
    i = 0
    queue.append(processes[0])
    i += 1
    while queue:
        current = queue.pop(0)
        if current.start_time is None:
            current.start_time = time
        run_time = min(quantum, remaining[current.pid])
        gantt.append((f"P{current.pid}", time, time + run_time, current.color))
        time += run_time
        remaining[current.pid] -= run_time
        if remaining[current.pid] == 0:
            current.finish_time = time
            current.turnaround_time = current.finish_time - current.arrival_time
```


```python
        gantt.append((f"P{current.pid}", time, time + run_time, current.color))
        time += run_time
        remaining[current.pid] -= run_time
        if remaining[current.pid] == 0:
            current.finish_time = time
            current.turnaround_time = current.finish_time - current.arrival_time
            current.waiting_time = current.turnaround_time - current.burst_time
            completed.append(current)
        else:
            while i < len(processes) and processes[i].arrival_time <= time:
                queue.append(processes[i])
                i += 1
            queue.append(current)
        if not queue and len(completed) < len(processes):
            while i < len(processes):
                if processes[i].arrival_time > time:
                    time = processes[i].arrival_time
                queue.append(processes[i])
                i += 1
                break
    return completed, gantt
```

Errors and Issues Faced During Development

During the development of the **CPU Scheduling Simulation Project using Python and Tkinter**, we encountered several technical and logical issues related to the implementation of scheduling algorithms and the graphical user interface. Below is a summary of the main problems we faced and how we resolved them:

---

1. Issue: Processes appeared duplicated or missing in the Gantt Chart

Cause: This happened because the original process list was being altered inside the algorithms, affecting the data across different runs.

Solution: We created a deep copy of the process list within each algorithm function to avoid modifying the original data. We also tracked the exact start and finish time of each process to maintain consistency in the Gantt Chart.

---

2. Issue: Round Robin algorithm not working correctly

Cause: The algorithm did not properly return processes to the end of the queue after each quantum, or it finished them prematurely.

Solution: We re-implemented the algorithm using a proper queue structure and added a variable to track the remaining burst time of each process. We ensured accurate quantum handling by updating the current time correctly.

---

3. Issue: Incorrect calculation of Waiting Time and Turnaround Time

Cause: These values were calculated without accurately considering the real start time of each process, especially in non-preemptive algorithms.

Solution: We added variables to store the exact start and finish time for each process, then calculated waiting and turnaround time based on these timestamps.

---

4. Issue: Program crashes when running the algorithm with no input

Cause: There was no check to ensure that processes had been added before running a scheduling algorithm.

Solution: We added a validation check to ensure the process list is not empty before execution. If no processes exist, a warning message is shown to the user.

---

5. Issue: Unorganized or inconsistent GUI layout

Cause: The GUI components were originally placed randomly using pack() which made the layout hard to manage.

Solution: We switched to using grid() and organized the interface inside Frame containers to ensure a clean and user-friendly design.

---

6. Issue: Manual input of processes was time-consuming

Solution: We added a button to generate 5 random processes automatically with random values for arrival time, burst time, and priority, to make testing faster and easier.

◇ Project Summary

This project is a desktop application that simulates CPU scheduling algorithms using Python and Tkinter. It allows users to visualize how different scheduling techniques manage process execution and calculate key performance metrics such as waiting time and turnaround time. The tool is educational and useful for computer science students to understand how operating systems schedule processes in real-world environments.

◇ Tools and Technologies Used

Python: Programming language used for logic and implementation

Tkinter: For designing the graphical user interface (GUI)

Matplotlib: For drawing the Gantt Chart

Thonny: The IDE used for development and testing

◇ Main Features

Automatically generates 5 random processes

Supports 4 scheduling algorithms: FCFS, SJF, Priority, Round Robin

Displays a detailed Gantt chart for visual understanding

Shows process statistics including start time, finish time, waiting time, and turnaround time

Simple and interactive GUI for ease of use

◇ What the Project Does

This project simulates how CPU scheduling algorithms work. It allows the user to generate random processes and choose a scheduling algorithm (FCFS, SJF, Priority, or Round Robin). When the algorithm is run, it displays a Gantt chart and calculates performance metrics like Start Time, Finish Time, Waiting Time, and Turnaround Time for each process. The goal is to help users visualize how each algorithm handles process execution.

◇ Conclusion

This project provided a practical and visual way to understand how different CPU scheduling algorithms work. By simulating process scheduling and displaying results in a clear and interactive interface, it helped deepen our knowledge of operating system concepts. We faced some technical challenges along the way, but solving them made the experience even more rewarding. Overall, this project was not only educational but also fun to build!

Thank you !