University of Science and Technology Chittagong (USTC)
Faculty of Science, Engineering & Technology
Department of Computer Science & Engineering

Assignment - 1

Course code : CSE 328

Course Title : Operating Systems Lab

Project Title : : "Dynamic CPU Scheduling Simulator"

Submitted by :

Abhilash Chowdhury
Unique ID : 0022210005101010
Roll No : 22010110
Reg No : 888
Semester : 6th
Batch : 38th
Dept : CSE

Submitted to:

Prianka Das
[Lecturer]
Department of CSE
FSET, USTC

Abstract

This project focuses on developing a Dynamic CPU Scheduling Simulator to analyze the performance of three fundamental CPU scheduling algorithms: First-Come, First-Serve (FCFS), Shortest Job First (SJF), and Priority Scheduling. By allowing users to input process details like burst time, arrival time, and priority, the simulator visualizes the process execution using Gantt charts and compares scheduling performance based on metrics like average waiting time, turnaround time, and CPU utilization.

Keywords

CPU Scheduling, FCFS, SJF, Priority Scheduling, Gantt Chart, Arrival Time, Burst Time, Priority, Completion Time, Waiting Time, Turnaround Time,CPU utilization, Report Generation, Python, matplotlib.

Introduction

CPU scheduling is a critical aspect of operating system design. It determines the order in which processes are executed by the CPU, significantly impacting system performance and resource utilization. This project aims to simulate three widely-used CPU scheduling algorithms—FCFS, SJF, and Priority Scheduling - providing a platform for visualization and comparative analysis of these algorithms under varying conditions.

Background

CPU scheduling algorithms play a pivotal role in managing processes efficiently in a multitasking environment. They help in optimizing CPU performance by minimizing waiting time, turnaround time, and maximizing CPU utilization. In this project, we use Python and the matplotlib library to implement and visualize these scheduling algorithms.

A. Scheduling Algorithms Implemented

- **First-Come, First-Serve (FCFS):**

A simple scheduling algorithm that executes processes in the order they arrive.

- **Shortest Job First (SJF):**

Selects the process with the shortest burst time for execution.

- **Priority Scheduling:**

Processes are scheduled based on their priority, where the process with the highest priority is executed first.

B. Simulator :

The simulator takes user input (process ID, arrival time, burst time, priority) and runs the processes using the chosen scheduling algorithm. It calculates key metrics like waiting time, turnaround time and CPU utilization.

C. Visualization :

The project uses Gantt charts to visually represent the process execution order for each algorithm. This helps users see how processes are scheduled and compare their execution times.

D. Comparative Analysis :

After running the algorithms, the simulator provides a comparison of the key metrics. This allows users to see which algorithm performs better based on waiting time, turnaround time and CPU utilization.

Project Evaluation

The Dynamic CPU Scheduling Simulator was built using Python to simulate the behavior of each algorithm. The project was divided into several key modules :

A. Input Interface

The interface allows users to input process details, including process ID, arrival time, burst time, and priority (for the priority scheduling algorithm). Each process is then scheduled based on the selected algorithm.

B. Visualization

For each algorithm, the simulator generates a Gantt chart, which graphically represents the order of process execution. Additionally, key metrics such as waiting time and turnaround time for each process are calculated and displayed.

C. Performance Metrics

Once the simulation for all three algorithms is complete, a report comparing the performance metrics is generated. This includes a side-by-side comparison of average waiting time, turnaround time, and CPU utilization, helping users understand the strengths and weaknesses of each algorithm.

Setup Environment

The project was developed using Python, with the following steps involved in setting up the environment:

- Install PyCharm.
- Install Python.
- Install the matplotlib library for generating Gantt charts.
- Use numpy for handling process data.

Code Implementation

The project involves implementing and simulating three CPU scheduling algorithms: FCFS, SJF, and Priority Scheduling. Below is a brief explanation of the major functions used for scheduling, calculating metrics, and visualizing the results.

A. Process Class

Process: Defines a class to represent each process, storing attributes like process ID, arrival time, burst time, priority, and calculated metrics such as waiting time and turnaround time.

B. First-Come, First-Serve (FCFS) Algorithm

fcfs(processes): Schedules processes based on their arrival times. It calculates the start time, completion time, and updates the current time for each process.

C. Shortest Job First (SJF) Algorithm

sjf(processes): Implements the non-preemptive Shortest Job First algorithm. It sorts the ready processes by burst time and schedules the shortest job first.

D. Priority Scheduling Algorithm

priority_scheduling(processes): Schedules processes based on their priority. Processes with higher priority (lower priority number) are scheduled first.

E. Performance Metrics Calculation

calculate_metrics(processes): Calculates turnaround time, waiting time, and CPU utilization for all processes based on their scheduling results.

F. Gantt Chart Visualization

visualize_detailed_gantt_chart(processes, algorithm_name): Generates a Gantt chart for each algorithm to visually represent the process execution timeline, highlighting waiting and execution times.

G. Report Generation

print_report(processes, algorithm_name, cpu_utilization): Prints a detailed report for each algorithm, showing process completion times, turnaround time, waiting times, average turnaround time, average waiting time and cpu utilization.

H. Main Function

main(): Handles user input, executes each scheduling algorithm, calculates performance metrics, generates Gantt charts, and prints comparison reports for all algorithms.

Critical Evaluation

The code successfully performed under various conditions, providing accurate results for process scheduling and metric calculation. The visualization component using Gantt charts offers a clear, intuitive representation of how the algorithms schedule tasks. Some challenges were faced with SJF's dynamic nature, as processes need to be continuously re-evaluated for their burst times, but the overall outcome met the project's objectives.

Conclusion

The Dynamic CPU Scheduling Simulator serves as a valuable tool for understanding and comparing different scheduling algorithms. Through visual aids like Gantt charts and the computation of key performance metrics, the project offers insights into the relative strengths and weaknesses of FCFS, SJF, and Priority Scheduling.

Acknowledgment

The project was completed with the guidance of Mrs. Prianka Das, whose support throughout the process was invaluable.

References

Geeks for Geeks : https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems-with-gantt-chart/

Geeks for Geeks : https://www.geeksforgeeks.org/short-note-on-gantt-chart/

ResearchGate : https://www.researchgate.net/publication/3878121

ChatGPT : https://chatgpt.com/

Appendix

Code :

```
import matplotlib.pyplot as plt

class Process:
    def _init_(self, pid, arrival_time, burst_time, priority=0):
        self.pid = pid
        self.arrival_time = arrival_time
        self.burst_time = burst_time
        self.priority = priority
        self.completion_time = 0
        self.turnaround_time = 0
        self.waiting_time = 0
        self.start_time = -1  # To record when the process actually starts
```

```
def fcfs(processes):
    processes.sort(key=lambda x: x.arrival_time)
    current_time = 0
    for process in processes:
        if current_time < process.arrival_time:
            current_time = process.arrival_time

        process.start_time = current_time
        process.completion_time = current_time + process.burst_time
        current_time += process.burst_time
    return processes

def sjf(processes):
    current_time = 0
    completed = []
    ready_queue = []

    while processes or ready_queue:
        # Move all processes that have arrived into the ready queue
        while processes and processes[0].arrival_time <= current_time:
            ready_queue.append(processes.pop(0))

        if ready_queue:
            # Select the process with the shortest burst time from the ready queue
            ready_queue.sort(key=lambda x: x.burst_time)  # Shortest Job First
            current_process = ready_queue.pop(0)

            if current_time < current_process.arrival_time:
                current_time = current_process.arrival_time

            current_process.start_time = current_time
            current_process.completion_time = current_time + current_process.burst_time
            current_time += current_process.burst_time
            completed.append(current_process)
        else:
            current_time += 1  # No process is ready, increment time

    return completed

def priority_scheduling(processes):
    current_time = 0
    completed = []
    ready_queue = []

    while processes or ready_queue:
        # Move all processes that have arrived into the ready queue
        while processes and processes[0].arrival_time <= current_time:
            ready_queue.append(processes.pop(0))
```

```
        if ready_queue:
            # Select the process with the highest priority from the ready queue
            ready_queue.sort(key=lambda x: x.priority)  # Lower number means higher priority
            current_process = ready_queue.pop(0)

            if current_time < current_process.arrival_time:
                current_time = current_process.arrival_time

            current_process.start_time = current_time
            current_process.completion_time = current_time + current_process.burst_time
            current_time += current_process.burst_time
            completed.append(current_process)
        else:
            current_time += 1  # No process is ready, increment time

    return completed

def calculate_metrics(processes):
    total_burst_time = 0
    total_idle_time = 0
    for process in processes:
        process.turnaround_time = process.completion_time - process.arrival_time
        process.waiting_time = process.turnaround_time - process.burst_time
        total_burst_time += process.burst_time
        total_idle_time += (process.start_time - process.arrival_time)

    # Calculate CPU utilization
    total_time = max(p.completion_time for p in processes) - min(p.arrival_time for p in processes)
    cpu_utilization = (total_burst_time / total_time) * 100 if total_time > 0 else 0

    return cpu_utilization

def visualize_detailed_gantt_chart(processes, algorithm_name):
    fig, gnt = plt.subplots()
    gnt.set_title(f"Gantt Chart for {algorithm_name} (With Waiting(Orange) & Turnaround
    Times(Orange+Blue))")
    gnt.set_xlabel('Time')
    gnt.set_ylabel('Process ID')

    max_time = max(p.completion_time for p in processes)
    gnt.set_xlim(0, max_time)
    gnt.set_ylim(0, len(processes))

    plt.xticks(range(0, max_time + 1))  # Show time ticks as integers

    for idx, process in enumerate(processes):
        # Plot waiting time (before start time)
        if process.start_time > process.arrival_time:
            gnt.broken_barh([(process.arrival_time, process.start_time - process.arrival_time)],
                            (idx, 1), facecolors=('tab:orange'))  # Waiting time in orange
```

```
        # Plot burst time (execution time)
        gnt.broken_barh([(process.start_time, process.burst_time)],
                        (idx, 1), facecolors=('tab:blue'))  # Execution time in blue

        plt.text(process.start_time + process.burst_time / 2, idx + 0.5, f'P{process.pid}', ha='center')

        plt.text(process.completion_time + 0.1, idx + 0.5,
            f' AT: {process.arrival_time}, BT: {process.burst_time}, \n\n TAT: 
{process.turnaround_time}, WT: {process.waiting_time}', va='center')

    plt.show()

def print_report(processes, algorithm_name, cpu_utilization):
    print(f"Report for {algorithm_name}")
    print(

f"{'PID':<5}{'Arrival':<10}{'Burst':<10}{'Priority':<10}{'Completion':<15}{'Turnaround':<15}{'W
aiting':<10}")
    for process in processes:

print(f"{process.pid:<5}{process.arrival_time:<10}{process.burst_time:<10}{process.priority:<10}
"

f"{process.completion_time:<15}{process.turnaround_time:<15}{process.waiting_time:<10}")
    avg_waiting_time = sum(p.waiting_time for p in processes) / len(processes)
    avg_turnaround_time = sum(p.turnaround_time for p in processes) / len(processes)
    print(f"Average Turnaround Time: {avg_turnaround_time}")
    print(f"Average Waiting Time: {avg_waiting_time}")
    print(f"CPU Utilization: {cpu_utilization:.2f}%")
    return avg_waiting_time, avg_turnaround_time, cpu_utilization


def get_user_input():
    processes = []
    num_processes = int(input("Enter the number of processes: "))

    for _ in range(num_processes):
        pid = int(input(f"\nEnter Process ID : "))
        arrival_time = int(input(f"Enter Arrival Time for process {pid}: "))
        burst_time = int(input(f"Enter Burst Time for process {pid}: "))
        priority = int(input(f"Enter Priority for process {pid} (Higher number means lower priority): "))

        processes.append(Process(pid, arrival_time, burst_time, priority))

    return processes


def main():
    processes = get_user_input()

    algorithms = {'FCFS': fcfs, 'SJF': sjf, 'Priority': priority_scheduling}
    comparison_data = {}
```

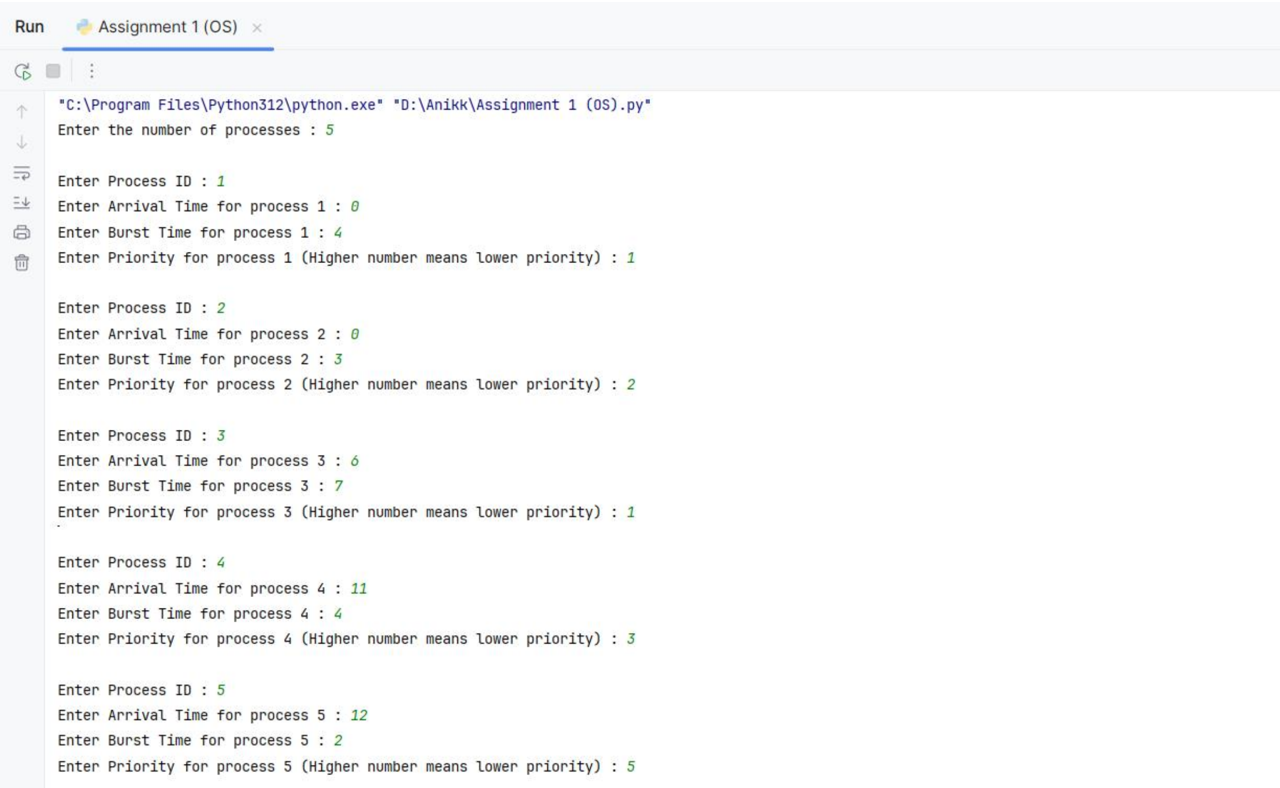
```
    for name, algorithm in algorithms.items():
        print(f"\nRunning {name} Scheduling")
        scheduled_processes = algorithm(processes.copy())
        calculate_metrics(scheduled_processes)
        cpu_utilization = calculate_metrics(scheduled_processes)
        visualize_detailed_gantt_chart(scheduled_processes, name)
        avg_waiting_time, avg_turnaround_time, cpu_utilization = print_report(scheduled_processes, name, cpu_utilization)

        comparison_data[name] = {
            'Average Waiting Time': avg_waiting_time,
            'Average Turnaround Time': avg_turnaround_time,
            'CPU Utilization': cpu_utilization
        }

    print("\n--------------------------- Comparison Report ---------------------------")
    print(f"{'Algorithm':<15}{'Avg Waiting Time':<20}{'Avg Turnaround Time':<20}{'CPU Utilization (%)':<20}")
    for algo, metrics in comparison_data.items():
        print(f"{algo:<15}{metrics['Average Waiting Time']:<20}{metrics['Average Turnaround Time']:<20}{metrics['CPU Utilization']:<20}")

if _name_ == '_main_':
  main()
```

Input Interface :


Run   Assignment 1 (OS)

```
"C:\Program Files\Python312\python.exe" "D:\Anikk\Assignment 1 (OS).py"
Enter the number of processes : 5

Enter Process ID : 1
Enter Arrival Time for process 1 : 0
Enter Burst Time for process 1 : 4
Enter Priority for process 1 (Higher number means lower priority) : 1

Enter Process ID : 2
Enter Arrival Time for process 2 : 0
Enter Burst Time for process 2 : 3
Enter Priority for process 2 (Higher number means lower priority) : 2

Enter Process ID : 3
Enter Arrival Time for process 3 : 6
Enter Burst Time for process 3 : 7
Enter Priority for process 3 (Higher number means lower priority) : 1

Enter Process ID : 4
Enter Arrival Time for process 4 : 11
Enter Burst Time for process 4 : 4
Enter Priority for process 4 (Higher number means lower priority) : 3

Enter Process ID : 5
Enter Arrival Time for process 5 : 12
Enter Burst Time for process 5 : 2
Enter Priority for process 5 (Higher number means lower priority) : 5
```

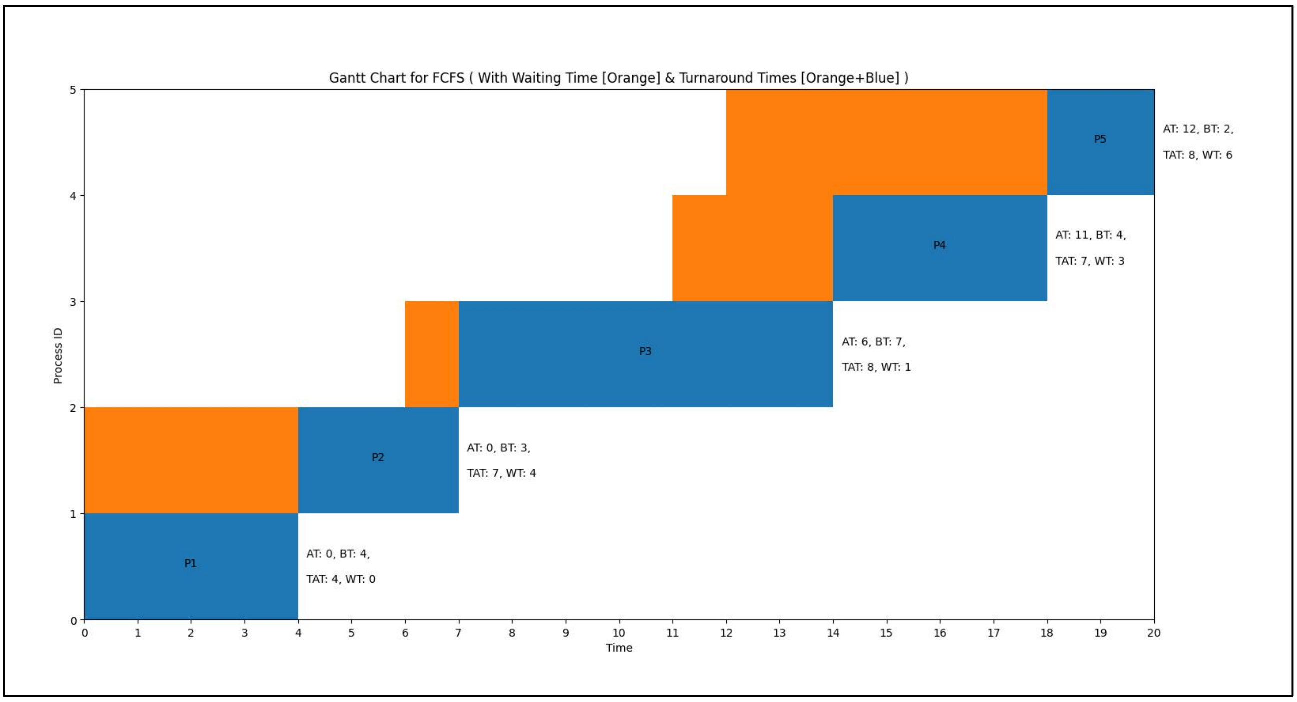
Simulation and Gantt Chart Visualization :

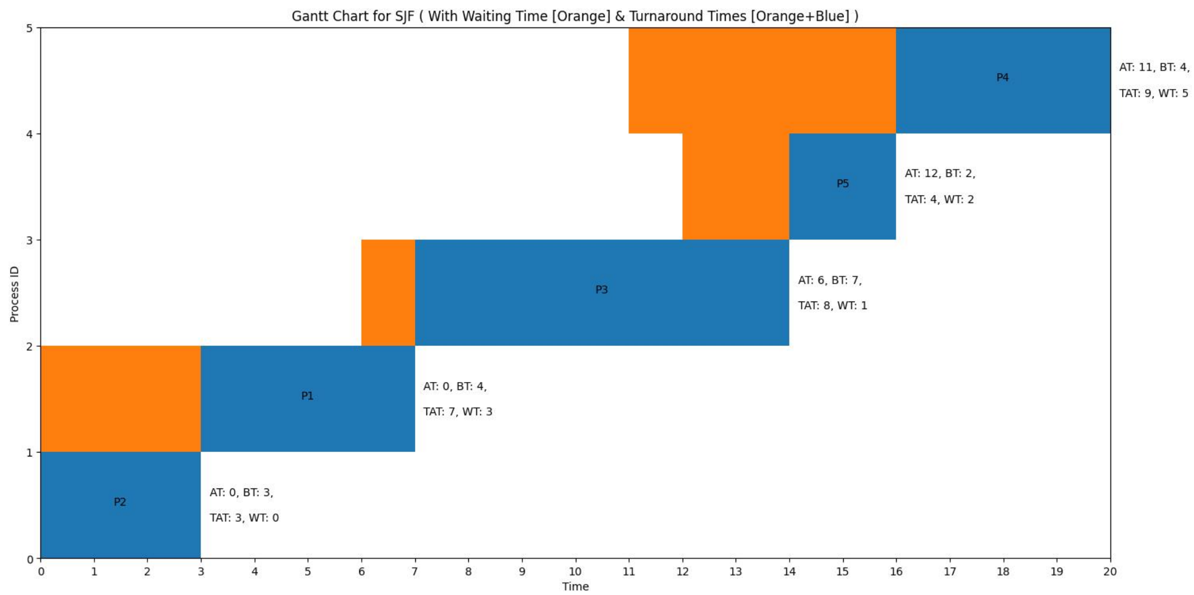
Run   Assignment 1 (OS)

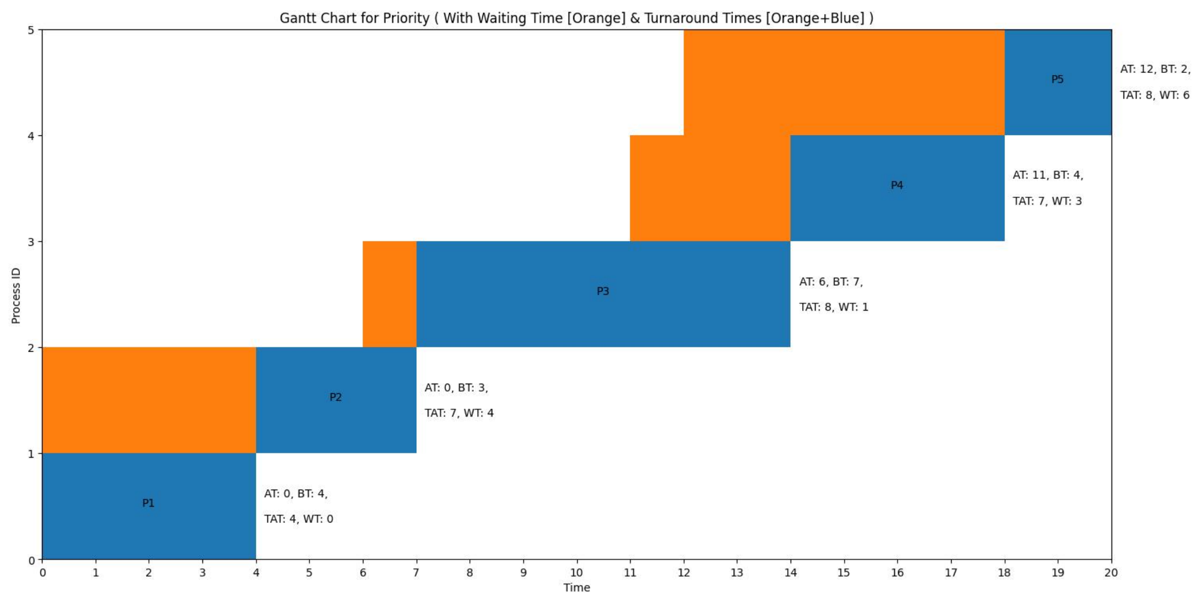
```
Running FCFS Scheduling
Report for FCFS
PID  Arrival   Burst     Priority  Completion     Turnaround     Waiting
1    0         4         1         4              4              0
2    0         3         2         7              7              4
3    6         7         1         14             8              1
4    11        4         3         18             7              3
5    12        2         5         20             8              6
Average Turnaround Time: 6.8
Average Waiting Time: 2.8
CPU Utilization: 100.00%

Running SJF Scheduling
Report for SJF
PID  Arrival   Burst     Priority  Completion     Turnaround     Waiting
2    0         3         2         3              3              0
1    0         4         1         7              7              3
3    6         7         1         14             8              1
5    12        2         5         16             4              2
4    11        4         3         20             9              5
Average Turnaround Time: 6.2
Average Waiting Time: 2.2
CPU Utilization: 100.00%

Running Priority Scheduling
Report for Priority
PID  Arrival   Burst     Priority  Completion     Turnaround     Waiting
1    0         4         1         4              4              0
2    0         3         2         7              7              4
3    6         7         1         14             8              1
4    11        4         3         18             7              3
5    12        2         5         20             8              6
Average Turnaround Time: 6.8
Average Waiting Time: 2.8
CPU Utilization: 100.00%
```







Comparison Report :

```
----------------------------- Comparison Report -----------------------------
Algorithm      Avg Waiting Time    Avg Turnaround Time CPU Utilization (%)
FCFS           2.8                 6.8                 100.0
SJF            2.2                 6.2                 100.0
Priority       2.8                 6.8                 100.0

Process finished with exit code 0
```