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

Assignment - 2

Course code : CSE 328

Course Title : Operating Systems Lab

Project Title : "Round Robin 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 implements and simulates the Round Robin Scheduling algorithm, demonstrating its efficacy in process scheduling. The algorithm is tested with a set of processes, each defined by specific burst and arrival times. Outputs include key performance metrics—turnaround and waiting times—as well as a Gantt chart visualizing process execution.

Keywords

Round Robin Scheduling, CPU Scheduling, Gantt Chart, Process Management.

Introduction

The Round Robin Scheduling algorithm is a preemptive CPU scheduling technique widely used in time-sharing systems. It assigns a fixed time quantum to each process and ensures fair CPU time allocation, reducing process starvation. In this project, a Python implementation of the algorithm is created to simulate the scheduling of three processes with predetermined burst and arrival times.

Background

A. Project :

The goal of the project is to simulate the Round Robin Scheduling algorithm, calculate essential metrics such as turnaround and waiting times, and visualize the scheduling order using a Gantt chart.

B. Algorithm Overview :

The algorithm works by executing processes in a cyclic manner, allocating a fixed time quantum to each process. If a process does not complete within its allocated quantum, it is preempted and re-added to the ready queue. This approach ensures an equitable distribution of CPU time..

Project Evaluation

A. Simulation Details

The following processes were used in the simulation:

Process ID	Burst Time (ms)	Arrival Time (ms)
P1	10	0
P2	5	1
P3	8	2

B. Calculations

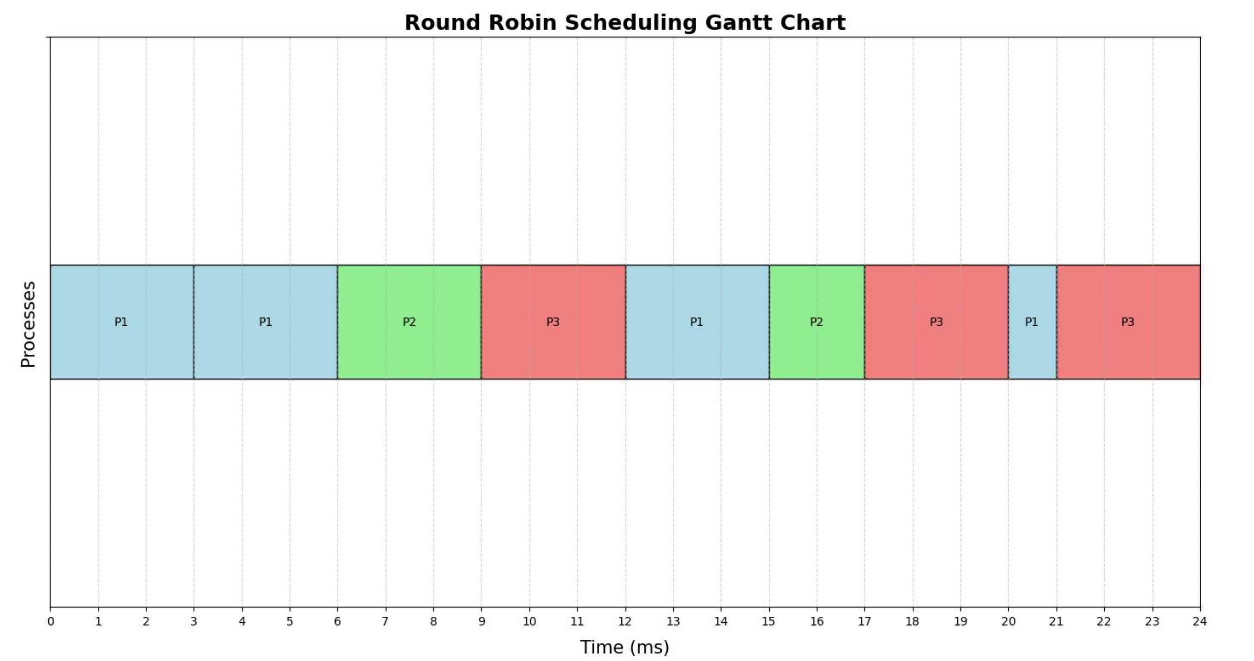
Turnaround Time (TAT) : Total time taken by each process from arrival to completion.

Waiting Time (WT) : Total time a process spends waiting in the ready queue.

```
Run    Ai

"C:\Program Files\Python312\python.exe" D:\Abhilash2\Ai.py
+---------+--------------+------------+-----------------+-----------------+--------------+
| Process | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time |
+---------+--------------+------------+-----------------+-----------------+--------------+
|   P1    |      0       |     10     |       21        |       21        |      11      |
|   P2    |      1       |     5      |       17        |       16        |      9       |
|   P3    |      2       |     8      |       23        |       21        |      12      |
+---------+--------------+------------+-----------------+-----------------+--------------+

Average Turnaround Time: 19.33 ms
Average Waiting Time: 10.67 ms

Gantt Chart: [('P1', 0), ('P1', 3), ('P2', 6), ('P3', 9), ('P1', 12), ('P2', 15), ('P3', 17), ('P1', 20), ('P3', 21)]

Process finished with exit code 0
```

B. Gantt Chart

The Gantt chart represents the order in which processes were executed :



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 to display process scheduling results (such as completion time, turnaround time, and waiting time) in a tabular format for better readability.
- Insatall the tabulate liabrary to display process scheduling results (such as completion time, turnaround time, and waiting time) in a tabular format for better readability.

Code Implementation

The implementation of the Round Robin Scheduling algorithm was carried out in Python. Below are the key points of the implementation :

Input Data Representation :

Processes were represented as tuples, each containing Process ID, Burst Time, and Arrival Time.
The time quantum for the scheduling algorithm was set to 3 milliseconds.
Algorithm Logic:

Initialization :

Data structures were initialized to store burst times, remaining burst times, completion times, turnaround times, and waiting times.

Ready Queue Management :

A ready queue was used to manage the order of execution of processes based on their arrival times.

Execution :

Each process was executed for the minimum of its remaining burst time or the time quantum. If the process did not finish execution within the time quantum, it was re-added to the ready queue.

Completion :

Processes were marked as complete when their remaining burst time reached zero, and their completion time was recorded.

Metric Calculation :

Turnaround Time (TAT) : Calculated as Completion Time - Arrival Time.

Waiting Time (WT) : Calculated as Turnaround Time - Burst Time.

Tabular Output :

The calculated metrics (Completion Time, Turnaround Time, Waiting Time) were displayed in a tabular format using the tabulate library for clarity and organization.

Average Metrics :

Average Turnaround Time and Average Waiting Time were calculated by taking the mean of their respective values for all processes.

Gantt Chart Visualization :

A Gantt chart was created using the matplotlib library to visually represent the scheduling order of processes over time. Each process execution was displayed as a color-coded bar, annotated with the process ID and its start and end times.

Setup and Execution :

The script was executed in a Python environment. The results, including the tabular metrics and the Gantt chart, were displayed as output.

Critical Evaluation

The simulation highlights the benefits of Round Robin Scheduling, such as fair CPU time allocation and reduced process starvation. However, the choice of time quantum significantly impacts performance. A small quantum increases overhead, while a large quantum leads to longer wait times.

Conclusion

The Round Robin Scheduling algorithm ensures fairness and efficiency in CPU scheduling. This project successfully demonstrates its application and visualizes the process scheduling order. By tweaking the time quantum, the algorithm can be optimized for various workloads.

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/round-robin-scheduling-with-different-arrival-times/

Scaler : https://www.scaler.com/topics/round-robin-scheduling-in-os/

ChatGPT : https://chatgpt.com/

Appendix

Code :

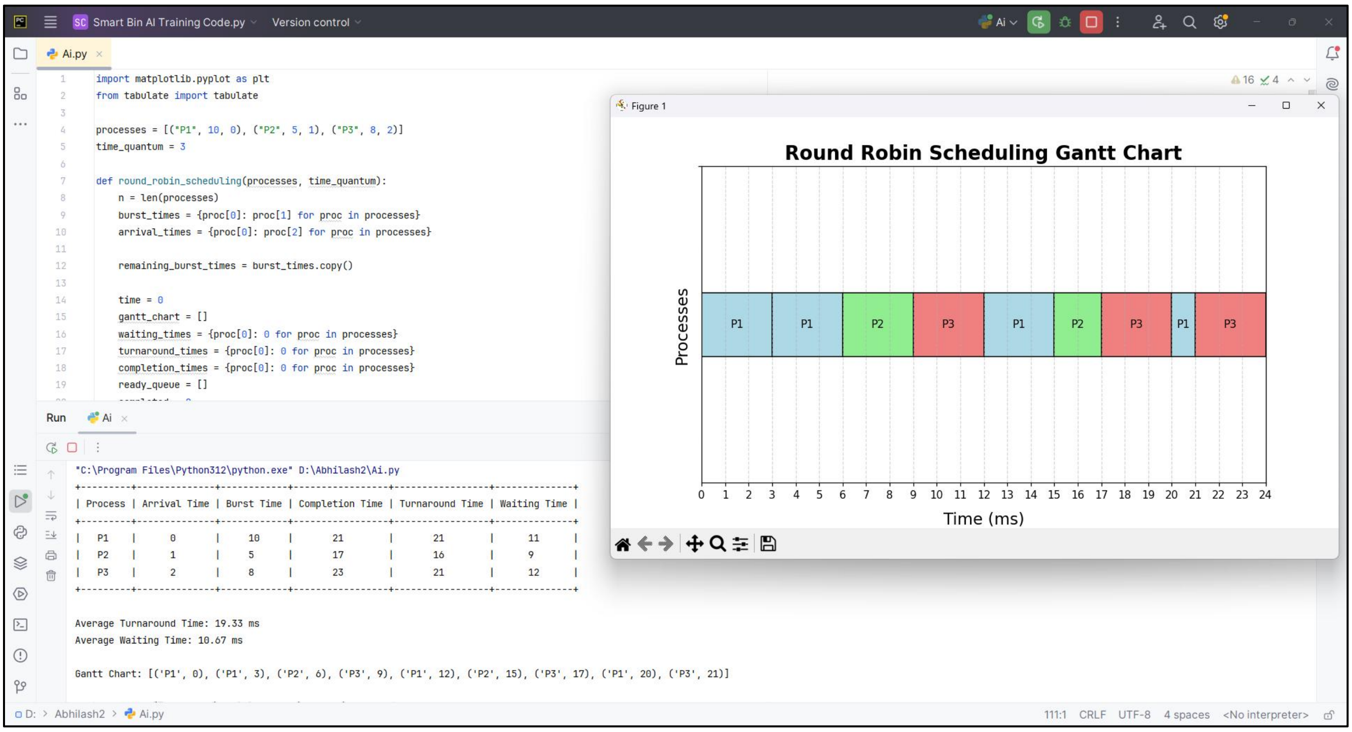
```
import matplotlib.pyplot as plt
from tabulate import tabulate

processes = [("P1", 10, 0), ("P2", 5, 1), ("P3", 8, 2)]
time_quantum = 3

def round_robin_scheduling(processes, time_quantum):
    n = len(processes)
    burst_times = {proc[0]: proc[1] for proc in processes}
    arrival_times = {proc[0]: proc[2] for proc in processes}

    remaining_burst_times = burst_times.copy()

    time = 0
    gantt_chart = []
    waiting_times = {proc[0]: 0 for proc in processes}
    turnaround_times = {proc[0]: 0 for proc in processes}
    completion_times = {proc[0]: 0 for proc in processes}
    ready_queue = []
    completed = 0

    while completed < n:
        for proc in processes:
            if proc[0] not in ready_queue and proc[2] <= time and remaining_burst_times[proc[0]] > 0:
                ready_queue.append(proc[0])

        if not ready_queue:
            time += 1
            continue

        current_proc = ready_queue.pop(0)
        gantt_chart.append((current_proc, time))
```

```
        execution_time = min(time_quantum, remaining_burst_times[current_proc])
        remaining_burst_times[current_proc] -= execution_time
        time += execution_time

        for proc in processes:
            if proc[0] in ready_queue:
                waiting_times[proc[0]] += execution_time

        if remaining_burst_times[current_proc] > 0:
            ready_queue.append(current_proc)
        else:
            completed += 1
            completion_times[current_proc] = time
            turnaround_times[current_proc] = time - arrival_times[current_proc]

    return gantt_chart, waiting_times, turnaround_times, completion_times

gantt_chart, waiting_times, turnaround_times, completion_times =
round_robin_scheduling(processes, time_quantum)

avg_tat = sum(turnaround_times.values()) / len(processes)
avg_wt = sum(waiting_times.values()) / len(processes)

table = []
for proc in processes:
    proc_id = proc[0]
    arrival_time = proc[2]
    burst_time = proc[1]
    completion_time = completion_times[proc_id]
    turnaround_time = turnaround_times[proc_id]
    waiting_time = waiting_times[proc_id]
    table.append([proc_id, arrival_time, burst_time, completion_time, turnaround_time,
waiting_time])

headers = ["Process", "Arrival Time", "Burst Time", "Completion Time", "Turnaround Time",
```

```
"Waiting Time"]
print(tabulate(table, headers=headers, tablefmt="pretty"))

print(f"\nAverage Turnaround Time: {avg_tat:.2f} ms")
print(f"Average Waiting Time: {avg_wt:.2f} ms")

print("\nGantt Chart:", gantt_chart)

def plot_gantt_chart(gantt_chart, time_quantum):
  fig, gnt = plt.subplots()
  gnt.set_ylim(0, 5)
  max_time = max(t[1] for t in gantt_chart) + 3
  gnt.set_xlim(0, max_time)
  gnt.set_xlabel('Time (ms)', fontsize=15,labelpad=10)
  gnt.set_ylabel('Processes', fontsize=15)

  x_ticks = range(0, max_time + 1, 1)
  gnt.set_xticks(x_ticks)

  yticks = [5]
  ylabels = ['']
  gnt.set_yticks(yticks)
  gnt.set_yticklabels(ylabels)

  colors = {"P1": "lightblue", "P2": "lightgreen", "P3": "lightcoral"}

  gnt.grid(True, axis='x', linestyle='--', alpha=0.5, zorder=0)

  for i, (proc, start_time) in enumerate(gantt_chart):
    if i < len(gantt_chart) - 1:
      end_time = gantt_chart[i + 1][1]
    else:
      end_time = start_time + time_quantum
    duration = end_time - start_time
```

```
        gnt.broken_barh([(start_time, duration)], (2, 1), facecolors=colors[proc], edgecolor='black')

        gnt.text(start_time + duration / 2, 2.5, proc, ha='center', va='center', color='black', fontsize=10, zorder=2)

    plt.title('Round Robin Scheduling Gantt Chart', fontsize=18, fontweight='bold')
    plt.show()

plot_gantt_chart(gantt_chart, time_quantum)
```

Output :




```
Run   Ai

"C:\Program Files\Python312\python.exe" D:\Abhilash2\Ai.py
+---------+--------------+------------+-----------------+-----------------+--------------+
| Process | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time |
+---------+--------------+------------+-----------------+-----------------+--------------+
|   P1    |      0       |     10     |       21        |       21        |      11      |
|   P2    |      1       |     5      |       17        |       16        |      9       |
|   P3    |      2       |     8      |       23        |       21        |      12      |
+---------+--------------+------------+-----------------+-----------------+--------------+

Average Turnaround Time: 19.33 ms
Average Waiting Time: 10.67 ms

Gantt Chart: [('P1', 0), ('P1', 3), ('P2', 6), ('P3', 9), ('P1', 12), ('P2', 15), ('P3', 17), ('P1', 20), ('P3', 21)]
```