| **Fr. Conceicao Rodrigues College of Engineering**  **Department of Computer Engineering** | | | |
| --- | --- | --- | --- |
| **Student’s Roll No** | **9913** | **Students Name** | **Mark lopes** |
| **Date of Performance** |  | **SE Computer – Div** | **A** |

**Aim:** Study Process Scheduling

**Lab Outcome:**

**CSL403.2:** Implement various Process scheduling algorithm and evaluate their performance.

**Problem Statements:**

Batch (A): First Come First Serve (FCFS) ,Non Preemptive Shortest Job First (SJF)

Batch (B): Non Preemptive Shortest Job First (SJF) ,Shortest Remaining Time First (SRTF)

Batch (C ): Round Robin Algorithm (RR), Non Preemptive Priority (NPP)

Batch (D): Non Preemptive Priority (NPP), Premptive Priority (PP)

1. Calculate WT, AWT, TAT, ATAT.

2. Compare the result of algorithms for a problem and find which algorithm is performing better.

**References:**

[**https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/?ref=lbp**](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/?ref=lbp)

| **On time Submission(2)** | **Knowledge of Topic(4)** | **Implementation and Demonstraion(4)** | **Total (10)** |
| --- | --- | --- | --- |
|  |  |  |  |
| **Signature of Faculty** |  | **Date of Submission** |  |

**Batch-C**

1.Round robin algorithm:-

emp\_list = [

{"employee": {"arrival\_time": 0, "burst\_time": 10, "completion\_time": -1, "turn\_around\_time": -1}},

{"employee": {"arrival\_time": 2, "burst\_time": 5, "completion\_time": -1, "turn\_around\_time": -1}},

{"employee": {"arrival\_time": 1, "burst\_time": 8, "completion\_time": -1, "turn\_around\_time": -1}}

]

time\_quantum = 2 # Time quantum for Round Robin

completed\_list = []

current\_time = 0

while emp\_list:

for emp in emp\_list:

if emp["employee"]["burst\_time"] > 0:

if emp["employee"]["burst\_time"] > time\_quantum:

current\_time += time\_quantum

emp["employee"]["burst\_time"] -= time\_quantum

else:

current\_time += emp["employee"]["burst\_time"] # Use remaining burst time

emp["employee"]["completion\_time"] = current\_time

emp["employee"]["turn\_around\_time"] = emp["employee"]["completion\_time"] - emp["employee"]["arrival\_time"]

emp["employee"]["burst\_time"] = 0

completed\_list.append(emp)

emp\_list.remove(emp)

break # Move to the next employee once the current one is completed

# Calculate AWT and ATAT

total\_waiting\_time = 0

total\_turnaround\_time = 0

for emp in completed\_list:

total\_waiting\_time += emp["employee"]["turn\_around\_time"] - emp["employee"]["burst\_time"]

total\_turnaround\_time += emp["employee"]["turn\_around\_time"]

# Calculate averages

total\_employees = len(completed\_list)

average\_waiting\_time = total\_waiting\_time / total\_employees

average\_turnaround\_time = total\_turnaround\_time / total\_employees

# Print completion times

for emp in completed\_list:

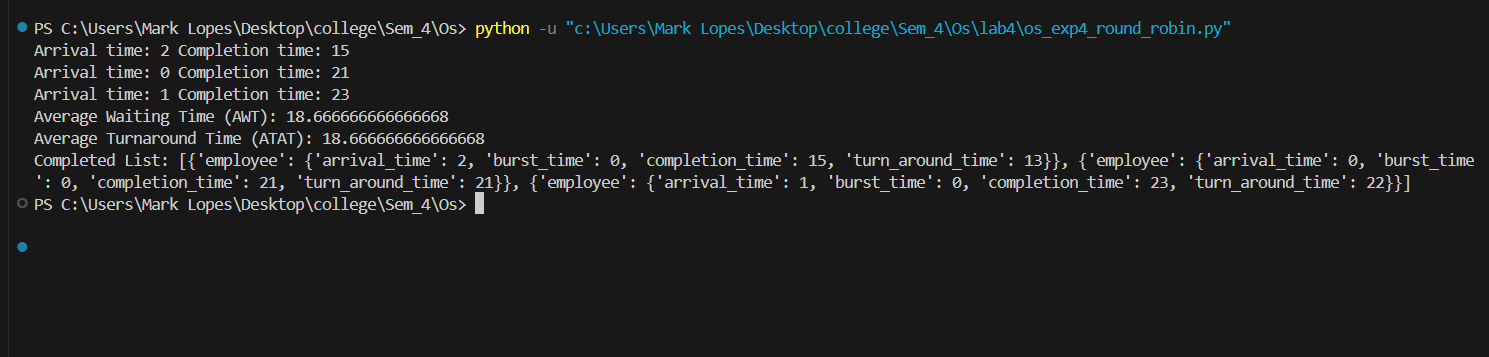
print("Arrival time:", emp["employee"]["arrival\_time"], "Completion time:", emp["employee"]["completion\_time"])

# Print AWT and ATAT

print("Average Waiting Time (AWT):", average\_waiting\_time)

print("Average Turnaround Time (ATAT):", average\_turnaround\_time)

print("Completed List:", completed\_list)



Non-Preemptive priority:-

#include <stdio.h>

struct Employee

{

int id;

char name[50];

int rank;

int arrivalTime;

int burstTime;

int waitingTime;

int turnaroundTime;

int completionTime; // New field to store completion time

};

void sortEmployeesByArrivalTime(struct Employee employees[], int n)

{

struct Employee temp;

for (int i = 0; i < n - 1; i++)

{

for (int j = 0; j < n - i - 1; j++)

{

// If arrival times are same, compare ranks

if (employees[j].arrivalTime == employees[j + 1].arrivalTime)

{

if (employees[j].rank > employees[j + 1].rank)

{

temp = employees[j];

employees[j] = employees[j + 1];

employees[j + 1] = temp;

}

}

// Otherwise, sort based on arrival time

else if (employees[j].arrivalTime > employees[j + 1].arrivalTime)

{

temp = employees[j];

employees[j] = employees[j + 1];

employees[j + 1] = temp;

}

}

}

}

void calculateTimes(struct Employee employees[], int n)

{

employees[0].waitingTime = 0;

employees[0].turnaroundTime = employees[0].burstTime;

employees[0].completionTime = employees[0].turnaroundTime + employees[0].arrivalTime;

for (int i = 1; i < n; i++)

{

employees[i].waitingTime = employees[i - 1].completionTime > employees[i].arrivalTime ? employees[i - 1].completionTime - employees[i].arrivalTime : 0;

employees[i].turnaroundTime = employees[i].waitingTime + employees[i].burstTime;

employees[i].completionTime = employees[i].turnaroundTime + employees[i].arrivalTime;

}

}

void displayMeetingSchedule(struct Employee employees[], int n)

{

printf("\nMeeting Schedule:\n");

printf("Employee\tRank\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\tCompletion Time\n");

for (int i = 0; i < n; i++)

{

printf("%s\t\t%d\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\n", employees[i].name, employees[i].rank,

employees[i].arrivalTime, employees[i].burstTime, employees[i].waitingTime,

employees[i].turnaroundTime, employees[i].completionTime);

}

}

int main(void)

{

int n;

printf("Enter the number of employees in the meeting: ");

scanf("%d", &n);

struct Employee employees[n];

for (int i = 0; i < n; i++)

{

employees[i].id = i + 1;

printf("Enter the name of Employee %d: ", i + 1);

scanf("%s", employees[i].name);

printf("Enter the rank (priority) for Employee %d: ", i + 1);

scanf("%d", &employees[i].rank);

printf("Enter the arrival time for Employee %d: ", i + 1);

scanf("%d", &employees[i].arrivalTime);

printf("Enter the burst time (time taken for presentation) for Employee %d: ", i + 1);

scanf("%d", &employees[i].burstTime);

}

// Sort employees by arrival time

sortEmployeesByArrivalTime(employees, n);

// Calculate waiting time, turnaround time, and completion time

calculateTimes(employees, n);

// Display meeting schedule

displayMeetingSchedule(employees, n);

// Calculate averages

float totalWaitingTime = 0;

float totalTurnaroundTime = 0;

for (int i = 0; i < n; i++)

{

totalWaitingTime += employees[i].waitingTime;

totalTurnaroundTime += employees[i].turnaroundTime;

}

float averageWaitingTime = totalWaitingTime / n;

float averageTurnaroundTime = totalTurnaroundTime / n;

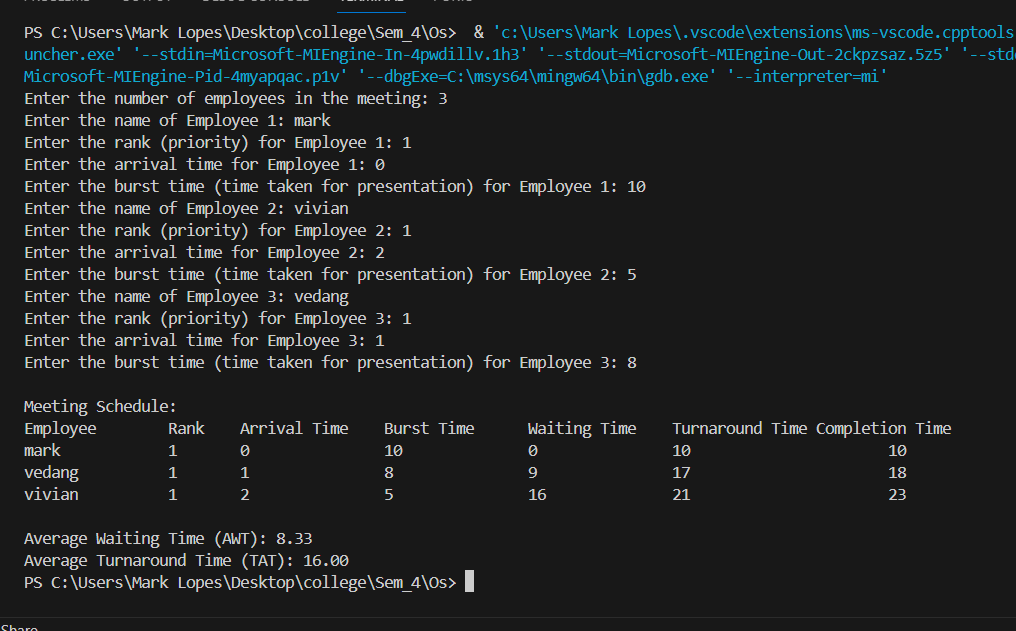
// Print averages

printf("\nAverage Waiting Time (AWT): %.2f\n", averageWaitingTime);

printf("Average Turnaround Time (TAT): %.2f\n", averageTurnaroundTime);

return 0;

}



As we see in the above results, the AWT and ATAT for non-preemptive priority scheduling algorithm is lower than that for round robin.

Therefore in our case for non-preemptive priority scheduling algorithm is better