End Term Project

On

Design Principles of Operating System (CSE 3249)

Submitted by

Name	**: Swayam Swarup Jethi**
Reg. No.	**: 2341010085**
Branch	**: CSE**
Semester	**: 5th**
Section	**: 2C1**
Session	**: 2025-2026**
Admission Batch	**: 2023**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

FACULTY OF ENGINEERING & TECHNOLOGY (ITER)

SIKSHA 'O' ANUSANDHAN DEEMED TO BE UNIVERSITY

BHUBANESWAR, ODISHA – 751030

PART – A

CPU Scheduling Algorithms

1.1. Problem Description

The objective of this part is to simulate CPU scheduling policies using a menu-driven C program.
The program implements:

1. First Come First Served (FCFS)
2. Round Robin (RR)

The scheduler selects processes from the ready queue based on the selected algorithm and displays the Gantt chart along with performance metrics.

1.2. Input Details

- Number of processes: 5
- Arrival time and burst time for each process
- Time quantum = 2 ms (for Round Robin)

1.3. Algorithm Used

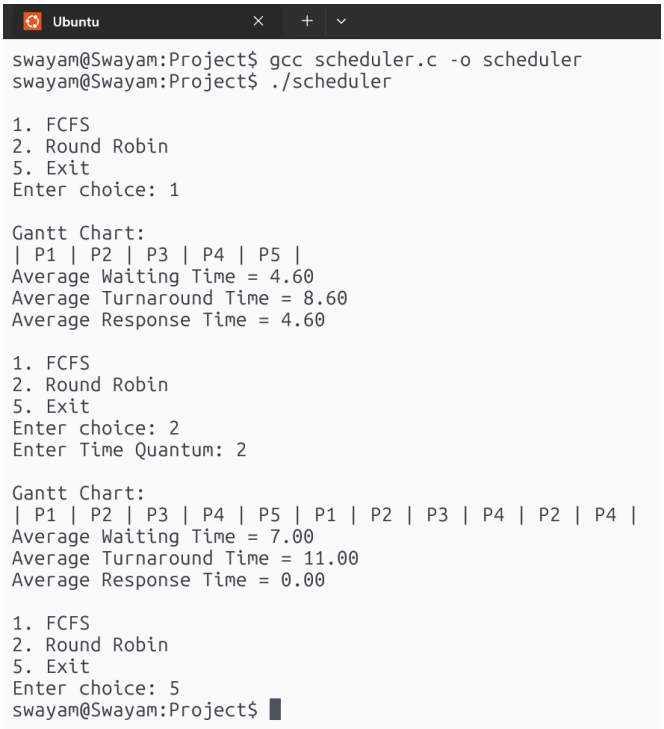
- **FCFS Scheduling Algorithm**
- **Round Robin Scheduling Algorithm**

1.4. Source Code (CPU Scheduling)

```c
#include <stdio.h>

typedef struct {
    int pid;
    int arrival;
    int burst;
    int remaining;
    int start;
    int completion;
    int waiting;
    int turnaround;
    int response;
    int started;
} Process;

void fcfs(Process p[], int n)
{
    int time = 0;
    float awt = 0, att = 0, art = 0;

    printf("\nGantt Chart:\n|");
    for (int i = 0; i < n; i++) {
        if (time < p[i].arrival)
            time = p[i].arrival;

        p[i].start = time;
        p[i].response = p[i].start - p[i].arrival;
        time += p[i].burst;
        p[i].completion = time;
        p[i].turnaround = p[i].completion - p[i].arrival;
        p[i].waiting = p[i].turnaround - p[i].burst;

        printf(" P%d |", p[i].pid);

        awt += p[i].waiting;
        att += p[i].turnaround;
        art += p[i].response;
    }

    printf("\nAverage Waiting Time = %.2f", awt / n);
    printf("\nAverage Turnaround Time = %.2f", att / n);
    printf("\nAverage Response Time = %.2f\n", art / n);
}

void roundRobin(Process p[], int n, int tq)
{
    int time = 0, completed = 0;
    float awt = 0, att = 0, art = 0;

    printf("\nGantt Chart:\n|");

    while (completed < n) {
        for (int i = 0; i < n; i++) {
            if (p[i].arrival <= time && p[i].remaining > 0) {

                if (!p[i].started) {
                    p[i].start = time;
                    p[i].response = p[i].start - p[i].arrival;
                    p[i].started = 1;
                }

                printf(" P%d |", p[i].pid);

                if (p[i].remaining > tq) {
                    time += tq;
                    p[i].remaining -= tq;
                } else {
                    time += p[i].remaining;
                    p[i].remaining = 0;
                    p[i].completion = time;
                    p[i].turnaround = p[i].completion - p[i].arrival;
                    p[i].waiting = p[i].turnaround - p[i].burst;

                    awt += p[i].waiting;
                    att += p[i].turnaround;
                    art += p[i].response;

                    completed++;
                }
            }
        }
    }

    printf("\nAverage Waiting Time = %.2f", awt / n);
    printf("\nAverage Turnaround Time = %.2f", att / n);
    printf("\nAverage Response Time = %.2f\n", art / n);
}
```

```c
int main()
{
    int n = 5, choice, tq;

    Process p[5] = {
        {1,0,3,3,0,0,0,0,0,0},
        {2,2,6,6,0,0,0,0,0,0},
        {3,4,4,4,0,0,0,0,0,0},
        {4,6,5,5,0,0,0,0,0,0},
        {5,8,2,2,0,0,0,0,0,0}
    };

    do {
        printf("\n1. FCFS\n2. Round Robin\n5. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            fcfs(p, n);
            break;

        case 2:
            printf("Enter Time Quantum: ");
            scanf("%d", &tq);
            roundRobin(p, n, tq);
            break;

        case 5:
            return 0;

        default:
            printf("Invalid choice\n");
        }
    } while (1);
}
```

1.5. Output



1.6. Analysis of Results

After comparing both algorithms, it is observed that **Round Robin scheduling provides better response time**, while **FCFS has simpler implementation but higher waiting time** for later processes.

PART – B

Banker's Algorithm (Deadlock Avoidance)

2.1. Problem Description

Banker's Algorithm is used to avoid deadlock by checking whether the system remains in a safe state after resource allocation.
The system consists of 5 processes and 4 types of resources.

2.2. Input Matrices

- Allocation Matrix
- Maximum Matrix
- Available Resources

2.3. Algorithm Used

- Banker's Algorithm for deadlock avoidance
- Safety Algorithm to determine safe sequence

2.4. Source Code (Banker's Algorithm)

Ubuntu

```
swayam@Swayam:Project$ cat banker.c
#include <stdio.h>

int main()
{
    int alloc[5][4] = {
        {0,0,1,2},
        {2,0,0,0},
        {0,0,3,4},
        {2,3,5,4},
        {0,3,3,2}
    };

    int max[5][4] = {
        {0,0,1,2},
        {2,7,5,0},
        {6,6,5,6},
        {4,3,5,6},
        {0,6,5,2}
    };

    int avail[4] = {2,1,0,0};
    int need[5][4], finish[5] = {0};
    int safeSeq[5], count = 0;

    for(int i=0;i<5;i++)
        for(int j=0;j<4;j++)
            need[i][j] = max[i][j] - alloc[i][j];

    printf("Need Matrix:\n");
    for(int i=0;i<5;i++){
        for(int j=0;j<4;j++)
            printf("%d ", need[i][j]);
        printf("\n");
    }
```

```
    while (count < 5) {
        int found = 0;
        for (int i = 0; i < 5; i++) {
            if (!finish[i]) {
                int j;
                for (j = 0; j < 4; j++)
                    if (need[i][j] > avail[j])
                        break;

                if (j == 4) {
                    for (int k = 0; k < 4; k++)
                        avail[k] += alloc[i][k];
                    safeSeq[count++] = i;
                    finish[i] = 1;
                    found = 1;
                }
            }
        }
        if (!found)
            break;
    }

    if (count == 5) {
        printf("\nSystem is in safe state.\nSafe Sequence: ");
        for (int i = 0; i < 5; i++)
            printf("P%d ", safeSeq[i] + 1);
    } else {
        printf("\nSystem is not in safe state.");
    }

    return 0;
}
^C
swayam@Swayam:Project$
```

2.5. Output

```
Ubuntu
swayam@Swayam:Project$ gcc banker.c -o banker
swayam@Swayam:Project$ ./banker
Need Matrix:
0 0 0 0
0 7 5 0
6 6 2 2
2 0 0 2
0 3 2 0

System is in safe state.
Safe Sequence: P1 P4 P5 P2 P3 swayam@Swayam:Project$
```

Conclusion

This project successfully demonstrates CPU scheduling algorithms and Banker's Algorithm.
The FCFS and Round Robin algorithms help in understanding CPU scheduling behavior, while Banker's Algorithm ensures deadlock-free resource allocation. Overall, the project provides practical insight into core operating system concepts.