Name – Harshit

MIS – 112316018

OS Lab

Assignment 5C

To write a C program for implementation of FCFS scheduling algorithms.

Code:

#include <stdio.h>

struct Process {

    int pid;

    int arrival\_time;

    int burst\_time;

    int completion\_time;

    int turnaround\_time;

    int waiting\_time;

};

void calculateTimes(struct Process processes[], int n) {

    processes[0].completion\_time = processes[0].arrival\_time + processes[0].burst\_time;

    processes[0].turnaround\_time = processes[0].completion\_time - processes[0].arrival\_time;

    processes[0].waiting\_time = processes[0].turnaround\_time - processes[0].burst\_time;

    for (int i = 1; i < n; i++) {

        if (processes[i].arrival\_time > processes[i-1].completion\_time) {

            processes[i].completion\_time = processes[i].arrival\_time + processes[i].burst\_time;

        } else {

            processes[i].completion\_time = processes[i-1].completion\_time + processes[i].burst\_time;

        }

        processes[i].turnaround\_time = processes[i].completion\_time - processes[i].arrival\_time;

        processes[i].waiting\_time = processes[i].turnaround\_time - processes[i].burst\_time;

    }

}

void displayProcesses(struct Process processes[], int n) {

    printf("\nProcess Execution Details:\n");

    printf("+---------+--------------+------------+----------------+----------------+--------------+\n");

    printf("| Process | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time |\n");

    printf("+---------+--------------+------------+----------------+----------------+--------------+\n");

    for (int i = 0; i < n; i++) {

        printf("| P%-6d | %-12d | %-10d | %-14d | %-14d | %-12d |\n",

               processes[i].pid,

               processes[i].arrival\_time,

               processes[i].burst\_time,

               processes[i].completion\_time,

               processes[i].turnaround\_time,

               processes[i].waiting\_time);

    }

    printf("+---------+--------------+------------+----------------+----------------+--------------+\n");

    float avg\_waiting\_time = 0, avg\_turnaround\_time = 0;

    for (int i = 0; i < n; i++) {

        avg\_waiting\_time += processes[i].waiting\_time;

        avg\_turnaround\_time += processes[i].turnaround\_time;

    }

    avg\_waiting\_time /= n;

    avg\_turnaround\_time /= n;

    printf("\nAverage Waiting Time = %.2f\n", avg\_waiting\_time);

    printf("Average Turnaround Time = %.2f\n", avg\_turnaround\_time);

}

void sortByArrivalTime(struct Process processes[], int n) {

    struct Process temp;

    for (int i = 0; i < n - 1; i++) {

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

            if (processes[j].arrival\_time > processes[j + 1].arrival\_time) {

                temp = processes[j];

                processes[j] = processes[j + 1];

                processes[j + 1] = temp;

            }

        }

    }

}

void displayGanttChart(struct Process processes[], int n) {

    int time\_scale = 2;

    int total\_burst\_time = 0, current\_time = 0;

    int timeline[100] = {0};

    int execution\_order[100] = {0};

    int execution\_count = 0;

    for (int i = 0; i < n; i++) {

        if (processes[i].arrival\_time > current\_time) {

            for (int j = current\_time; j < processes[i].arrival\_time; j++) {

                timeline[execution\_count] = 1;

                execution\_order[execution\_count] = 0;

                execution\_count++;

            }

            current\_time = processes[i].arrival\_time;

        }

        for (int j = 0; j < processes[i].burst\_time; j++) {

            timeline[execution\_count] = 1;

            execution\_order[execution\_count] = processes[i].pid;

            execution\_count++;

        }

        current\_time += processes[i].burst\_time;

    }

    printf("\nGantt Chart:\n");

    printf(" ");

    for (int i = 0; i < execution\_count; i++) {

        for (int j = 0; j < time\_scale; j++) {

            printf("-");

        }

        printf(" ");

    }

    printf("\n|");

    for (int i = 0; i < execution\_count; i++) {

        if (execution\_order[i] == 0) {

            for (int j = 0; j < time\_scale - 1; j++) {

                printf(" ");

            }

            printf("X");

        } else {

            for (int j = 0; j < time\_scale - 1; j++) {

                printf(" ");

            }

            printf("%d", execution\_order[i]);

        }

        printf("|");

    }

    printf("\n ");

    for (int i = 0; i < execution\_count; i++) {

        for (int j = 0; j < time\_scale; j++) {

            printf("-");

        }

        printf(" ");

    }

    printf("\n");

    printf("0");

    current\_time = 0;

    for (int i = 0; i < execution\_count; i++) {

        for (int j = 0; j < time\_scale; j++) {

            printf(" ");

        }

        current\_time += timeline[i];

        if (current\_time < 10) {

            printf("%d", current\_time);

        } else {

            printf("%d", current\_time);

        }

    }

    printf("\n");

    printf("\nNote: 'X' indicates CPU idle time\n");

}

int main() {

    int n;

    printf("Enter the number of processes: ");

    scanf("%d", &n);

    struct Process processes[n];

    for (int i = 0; i < n; i++) {

        processes[i].pid = i + 1;

        printf("\nEnter details for Process P%d:\n", i + 1);

        printf("Arrival Time: ");

        scanf("%d", &processes[i].arrival\_time);

        printf("Burst Time: ");

        scanf("%d", &processes[i].burst\_time);

    }

    sortByArrivalTime(processes, n);

    calculateTimes(processes, n);

    displayProcesses(processes, n);

    displayGanttChart(processes, n);

    return 0;

}

Output:

