Name – Harshit

MIS – 112316018

OS lab

Assignment 5D

To write a C program for implementation of SJF 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;

    int is\_completed;

};

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

    int current\_time = 0;

    int completed = 0;

    while (completed != n) {

        int idx = -1;

        int min\_burst = 9999;

        for (int i = 0; i < n; i++) {

            if (processes[i].arrival\_time <= current\_time && processes[i].is\_completed == 0) {

                if (processes[i].burst\_time < min\_burst) {

                    min\_burst = processes[i].burst\_time;

                    idx = i;

                }

                if (processes[i].burst\_time == min\_burst) {

                    if (processes[i].arrival\_time < processes[idx].arrival\_time) {

                        min\_burst = processes[i].burst\_time;

                        idx = i;

                    }

                }

            }

        }

        if (idx != -1) {

            processes[idx].completion\_time = current\_time + processes[idx].burst\_time;

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

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

            processes[idx].is\_completed = 1;

            completed++;

            current\_time = processes[idx].completion\_time;

        } else {

            current\_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 displayGanttChart(struct Process processes[], int n) {

    int i, j;

    int total\_burst\_time = 0;

    int current\_time = 0;

    int completed = 0;

    int execution\_order[100];

    int execution\_time[100];

    int count = 0;

    struct Process temp\_processes[n];

    for (i = 0; i < n; i++) {

        temp\_processes[i] = processes[i];

        temp\_processes[i].is\_completed = 0;

    }

    while (completed != n) {

        int idx = -1;

        int min\_burst = 9999;

        for (i = 0; i < n; i++) {

            if (temp\_processes[i].arrival\_time <= current\_time && temp\_processes[i].is\_completed == 0) {

                if (temp\_processes[i].burst\_time < min\_burst) {

                    min\_burst = temp\_processes[i].burst\_time;

                    idx = i;

                }

                if (temp\_processes[i].burst\_time == min\_burst) {

                    if (temp\_processes[i].arrival\_time < temp\_processes[idx].arrival\_time) {

                        min\_burst = temp\_processes[i].burst\_time;

                        idx = i;

                    }

                }

            }

        }

        if (idx != -1) {

            execution\_order[count] = temp\_processes[idx].pid;

            execution\_time[count] = current\_time;

            current\_time += temp\_processes[idx].burst\_time;

            temp\_processes[idx].is\_completed = 1;

            completed++;

            count++;

        } else {

            execution\_order[count] = 0; // Idle time

            execution\_time[count] = current\_time;

            current\_time++;

            count++;

        }

    }

    execution\_time[count] = current\_time; // Final time

    printf("\nGantt Chart:\n");

    printf(" ");

    for (i = 0; i < count; i++) {

        for (j = 0; j < execution\_time[i+1] - execution\_time[i]; j++) {

            printf("-");

        }

        printf(" ");

    }

    printf("\n|");

    for (i = 0; i < count; i++) {

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

            for (j = 0; j < execution\_time[i+1] - execution\_time[i] - 1; j++) {

                printf(" ");

            }

            printf("X");

        } else {

            for (j = 0; j < execution\_time[i+1] - execution\_time[i] - 1; j++) {

                printf(" ");

            }

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

        }

        printf("|");

    }

    printf("\n ");

    for (i = 0; i < count; i++) {

        for (j = 0; j < execution\_time[i+1] - execution\_time[i]; j++) {

            printf("-");

        }

        printf(" ");

    }

    printf("\n");

    for (i = 0; i <= count; i++) {

        if (execution\_time[i] < 10) {

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

        } else {

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

        }

        if (i < count) {

            for (j = 0; j < execution\_time[i+1] - execution\_time[i]; j++) {

                printf(" ");

            }

        }

    }

    printf("\n\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);

        processes[i].is\_completed = 0;

    }

    calculateTimesSJF(processes, n);

    displayProcesses(processes, n);

    displayGanttChart(processes, n);

    return 0;

}

Output:  
