6- C program to implement SRTF cpu scheduling algorithm.

**Shortest Remaining Time First (SRTF)** is a CPU scheduling algorithm used in operating systems to efficiently schedule processes. It is also known as **Preemptive Shortest Job Next (SJN)** It is a preemptive algorithm that selects the process with the smallest remaining burst time to execute next.

code-

#include <stdio.h>

*struct* process {

*int* pid;

*int* arrival\_time;

*int* burst\_time;

*int* start\_time;

*int* completion\_time;

*int* turnaround\_time;

*int* waiting\_time;

};

*int* main() {

*int* n;

*struct* process p[100];

*float* avg\_turnaround\_time;

*float* avg\_waiting\_time;

*float* cpu\_utilisation;

*int* total\_turnaround\_time = 0;

*int* total\_waiting\_time = 0;

*int* total\_idle\_time = 0;

*float* throughput;

*int* burst\_remaining[100];

*int* is\_completed[100];

    for (*int* i = 0; i < 100; i++) {

        is\_completed[i] = 0;

    }

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

    scanf("%d", &n);

    for (*int* i = 0; i < n; i++) {

        printf("enter the arrival time and burst time of %d process: ",i+1);

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

        p[i].pid = i + 1;

        burst\_remaining[i] = p[i].burst\_time;

    }

*int* current\_time = 0;

*int* completed = 0;

*int* prev = 0;

    while (completed != n) {

*int* idx = -1;

*int* mn = 10000000;

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

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

                if (burst\_remaining[i] < mn) {

                    mn = burst\_remaining[i];

                    idx = i;

                }

                if (burst\_remaining[i] == mn) {

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

                        mn = burst\_remaining[i];

                        idx = i;

                    }

                }

            }

        }

        if (idx != -1) {

            if (burst\_remaining[idx] == p[idx].burst\_time) {

                p[idx].start\_time = current\_time;

                total\_idle\_time += p[idx].start\_time - prev;

            }

            burst\_remaining[idx] -= 1;

            current\_time++;

            prev = current\_time;

            if (burst\_remaining[idx] == 0) {

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

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

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

                total\_turnaround\_time += p[idx].turnaround\_time;

                total\_waiting\_time += p[idx].waiting\_time;

                is\_completed[idx] = 1;

                completed++;

            }

        } else {

            current\_time++;

        }

    }

*int* min\_arrival\_time = 10000000;

*int* max\_completion\_time = -1;

    for (*int* i = 0; i < n; i++) {

        min\_arrival\_time = (p[i].arrival\_time < min\_arrival\_time) ? p[i].arrival\_time : min\_arrival\_time;

        max\_completion\_time = (p[i].completion\_time > max\_completion\_time) ? p[i].completion\_time : max\_completion\_time;

    }

    avg\_turnaround\_time = (*float*)total\_turnaround\_time / n;

    avg\_waiting\_time = (*float*)total\_waiting\_time / n;

    cpu\_utilisation = ((*float*)(max\_completion\_time - total\_idle\_time) / max\_completion\_time) \* 100;

    throughput = (*float*)n / (max\_completion\_time - min\_arrival\_time);

    printf("\nPid\tAT\tBT\tST\tCT\tTAT\tWT\n");

    for (*int* i = 0; i < n; i++) {

        printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrival\_time, p[i].burst\_time, p[i].start\_time, p[i].completion\_time, p[i].turnaround\_time, p[i].waiting\_time);

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

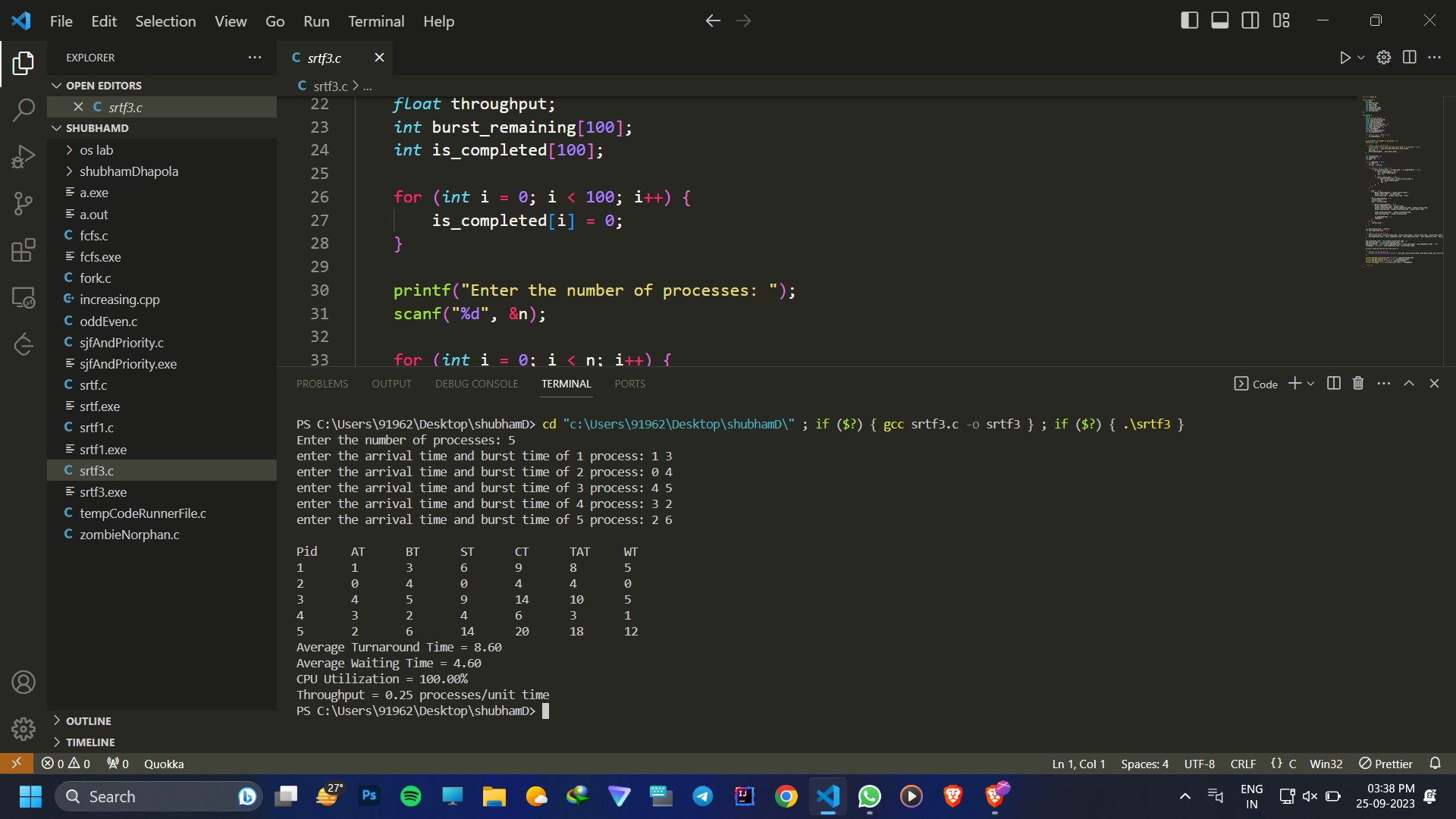
    printf("Average Waiting Time = %.2f\n", avg\_waiting\_time);

    printf("CPU Utilization = %.2f%%\n", cpu\_utilisation);

    printf("Throughput = %.2f processes/unit time\n", throughput);

    return 0;

}



A screenshot of a computer screen

Description automatically generated