1- C program to demonstrate the working of fork() system call.

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

*int* main()

{

    if(fork()==0)

    {

        printf("child process\n");

        printf("%d-pid %d-ppid\n",getpid(),getppid());

    }

    else

    {

        wait(NULL);

        printf("parent process\n");

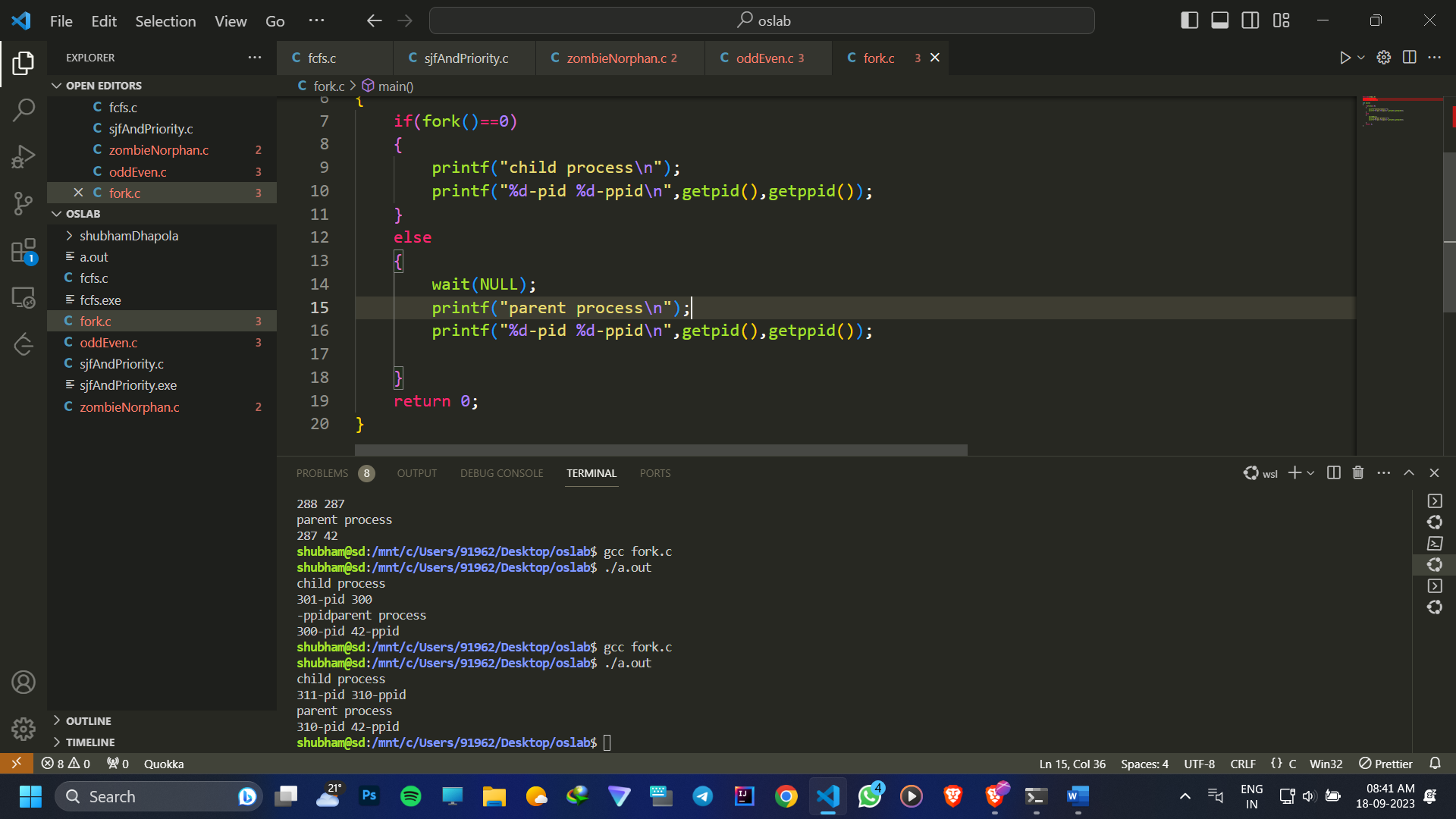
        printf("%d-pid %d-ppid\n",getpid(),getppid());

    }

    return 0;

}

Output:



2- C program in which parent process does sum of odd numbers and the child process does the sum of even numbers using fork() system call.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

*int* main()

{

*int* n , evenSum = 0, oddSum = 0;

    printf("Enter a positive integer: ");

    scanf("%d", &n);

    if (fork())

    {

        // Parent process

        for (*int* i = 1; i <= n; i += 2)

        {

            oddSum += i;

        }

         wait(NULL); // Wait for the child process to finish

        printf("Total odd sum: %d\n", oddSum);

    }

    else

    {

        // Child process

        for (*int* i = 2; i <= n; i += 2)

        {

            evenSum += i;

        }

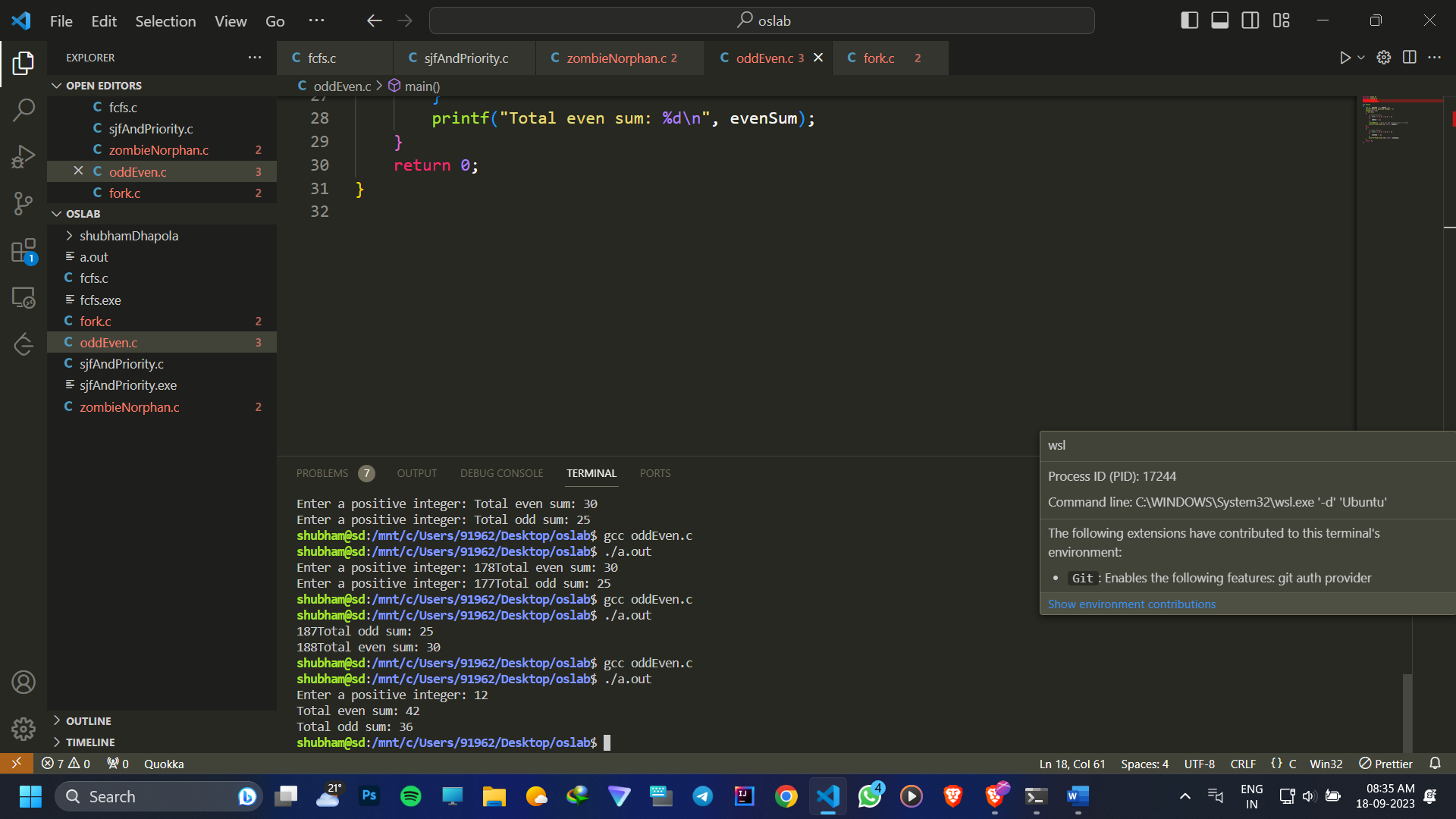
        printf("Total even sum: %d\n", evenSum);

    }

    return 0;

}

Output:



3- C program to implement orphan and zombie process.

#include <stdlib.h>

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

*int* main()

{

*int* choice;

    printf("1 for zombie or 2 for orphan process ");

    scanf("%d", &choice);

    // zombie process

    if (choice == 1)

    {

*pid\_t* child\_pid = fork();

        // parent process

        if (child\_pid > 0)

        {

            sleep(2);

            printf("this is parent process --zombie\n");

        }

        // Child process

        else if (child\_pid == 0)

            printf("this is child process\n");

    }

    // orphan process

    if (choice == 2)

    {

*int* pid = fork();

        // parent process

        if (pid > 0)

            printf("in parent process\n");

        else if (pid == 0)

        { // child process

            sleep(2);

            printf("in child process--orphan\n");

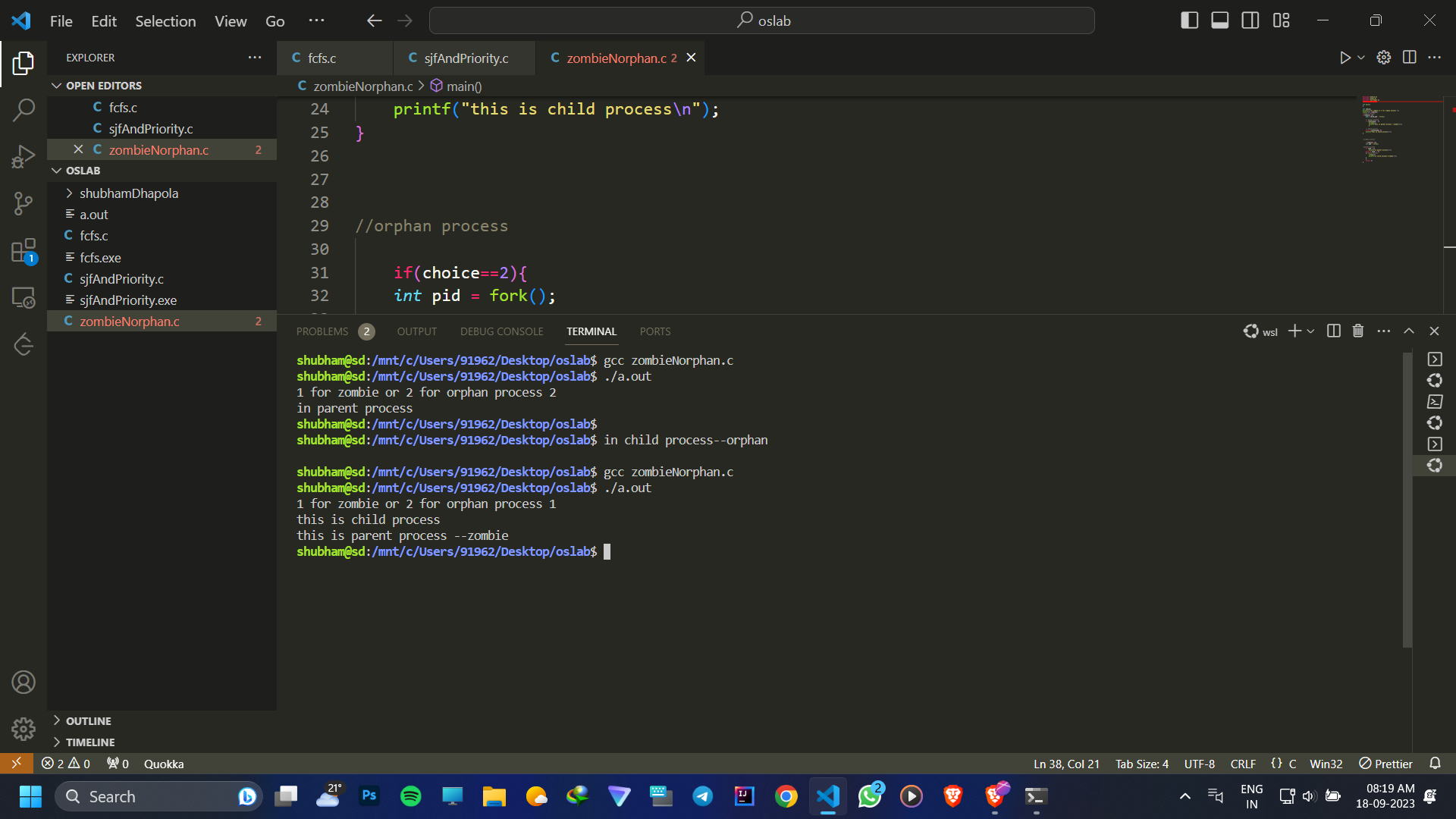
        }

    }

    return 0;

}

Output:



4- C program to implement FCFS CPU scheduling algorithm.

#include <stdio.h>

#include <stdlib.h>

*struct* Process {

*int* pid, arrival, burst, completion, turnaround, waiting;

};

*int* main() {

*int* n, total = 0, idle = 0;

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

    printf("Enter number of processes: ");

    scanf("%d", &n);

*struct* Process \*p = malloc(sizeof(*struct* Process) \* n);

    for (*int* i = 0; i < n; i++) {

        printf("Enter arrival time and burst time for process %d: ", i + 1);

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

        p[i].pid = i + 1;

    }

    for (*int* i = 0; i < n; i++) {

        p[i].completion = (i == 0) ? p[i].arrival + p[i].burst : p[i - 1].completion + p[i].burst;

        p[i].turnaround = p[i].completion - p[i].arrival;

        p[i].waiting = p[i].turnaround - p[i].burst;

        avg\_waiting += p[i].waiting;

        avg\_turnaround += p[i].turnaround;

        if (p[i].arrival > p[i - 1].completion)

            idle += p[i].arrival - p[i - 1].completion;

    }

    printf("\nPID\tArrival time \tBurst time\tCompletion time\tTurnaround time\tWaiting time\n");

    for (*int* i = 0; i < n; i++) {

        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i].pid, p[i].arrival, p[i].burst,

               p[i].completion, p[i].turnaround, p[i].waiting);

    avg\_waiting /= n;

    avg\_turnaround /= n;

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

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

    printf("Throughput: %.2f\n", (*float*)n / p[n - 1].completion);

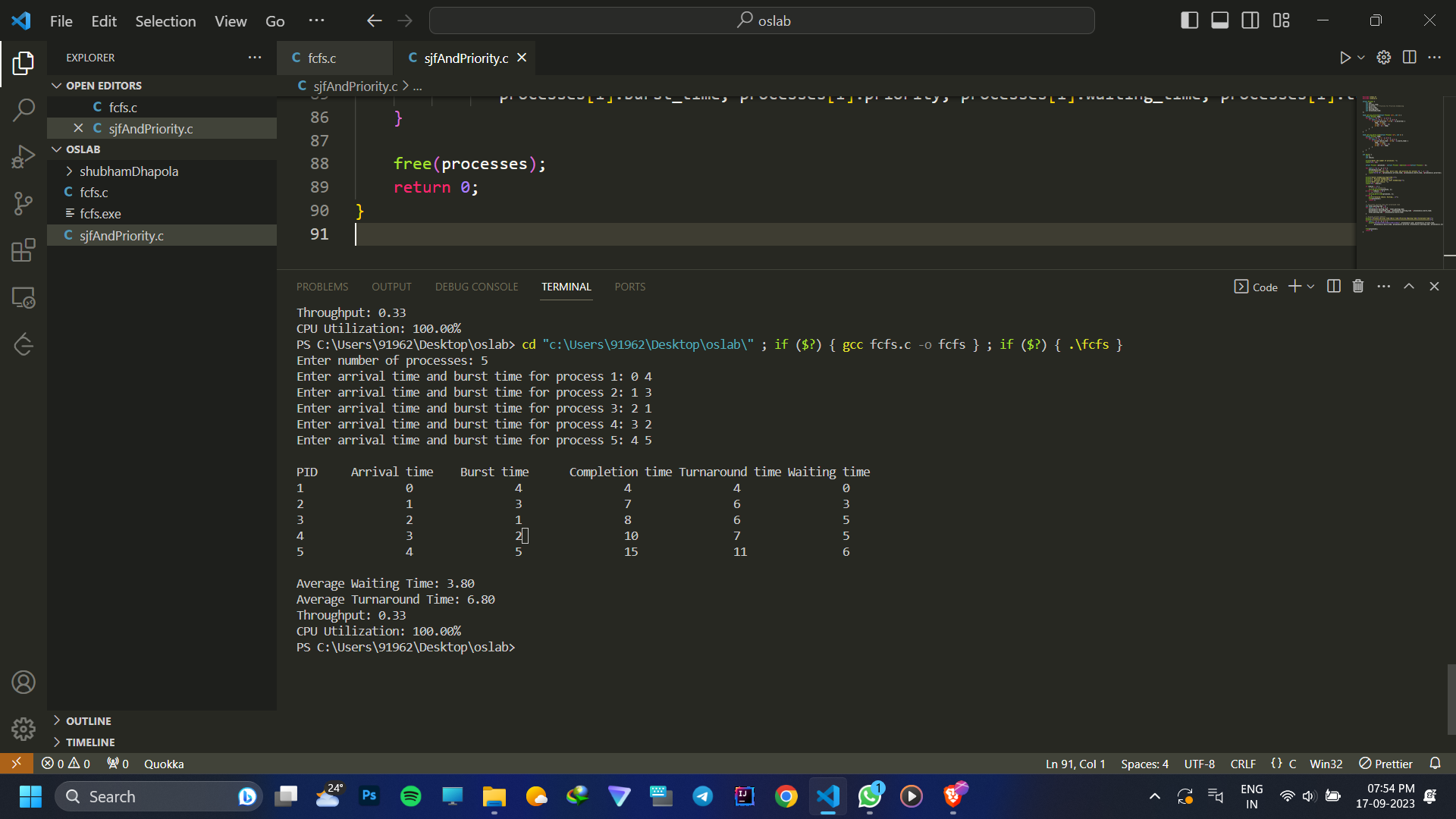
    printf("CPU Utilization: %.2f%%\n", (*float*)(p[n - 1].completion - idle) / p[n - 1].completion \* 100);

    free(p);

    return 0;

}

Output:



5- C program to implement SJF and Priority CPU scheduling algorithms.

#include <stdio.h>

#include <stdlib.h>

*struct* Process {

*int* pid;

*int* arrival;

*int* priority;  // Priority for Priority Scheduling

*int* burst;

*int* waiting;

*int* turnaround;

*int* completion;

};

*void* sort\_by\_priority(*struct* Process \**arr*, *int* *n*) {

*struct* Process temp;

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

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

            if (*arr*[j].priority > *arr*[j + 1].priority) {

                temp = *arr*[j];

*arr*[j] = *arr*[j + 1];

*arr*[j + 1] = temp;

            }

        }

    }

}

*void* sort\_by\_burst\_time(*struct* Process \**arr*, *int* *n*) {

*struct* Process temp;

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

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

            if (*arr*[j].burst > *arr*[j + 1].burst) {

                temp = *arr*[j];

*arr*[j] = *arr*[j + 1];

*arr*[j + 1] = temp;

            }

        }

    }

}

*int* main() {

*int* n;

*int* choice;

*int* total=0, idle=0;

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

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

    scanf("%d", &n);

*struct* Process \*p = (*struct* Process \*)malloc(sizeof(*struct* Process) \* n);

    for (*int* i = 0; i < n; i++) {

        p[i].pid = i + 1;

        printf("Enter arrival time, burst time, and priority for process %d: ", i + 1);

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

    }

    printf("Select Scheduling Algorithm:\n");

    printf("1. Priority Scheduling\n");

    printf("2. Shortest Job First (SJF) Scheduling\n");

    printf("Enter your choice: ");

    scanf("%d", &choice);

    if (choice == 1) {

        // Priority Scheduling

        sort\_by\_priority(p, n);

    } else if (choice == 2) {

        // SJF Scheduling

        sort\_by\_burst\_time(p, n);

    } else {

        printf("Invalid choice. Exiting...\n");

        free(p);

        return 1;

    }

     for (*int* i = 0; i < n; i++) {

        p[i].completion = (i == 0) ? p[i].arrival + p[i].burst : p[i - 1].completion + p[i].burst;

        p[i].turnaround = p[i].completion - p[i].arrival;

        p[i].waiting = p[i].turnaround - p[i].burst;

        avg\_waiting += p[i].waiting;

        avg\_turnaround += p[i].turnaround;

        if (p[i].arrival > p[i - 1].completion)

            idle += p[i].arrival - p[i - 1].completion;

    }

    printf("\nPID\tPriority\tBurst time\tCompletion time\tTurnaround time\tWaiting time\n");

    for (*int* i = 0; i < n; i++) {

        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i].pid, p[i].priority, p[i].burst,

               p[i].completion, p[i].turnaround, p[i].waiting);

    }

    avg\_waiting /= n;

    avg\_turnaround /= n;

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

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

    printf("Throughput: %.2f\n", (*float*)n / p[n - 1].completion);

    printf("CPU Utilization: %.2f%%\n", (*float*)(p[n - 1].completion - idle) / p[n - 1].completion \* 100);

    free(p);

    return 0;

}

Output:

