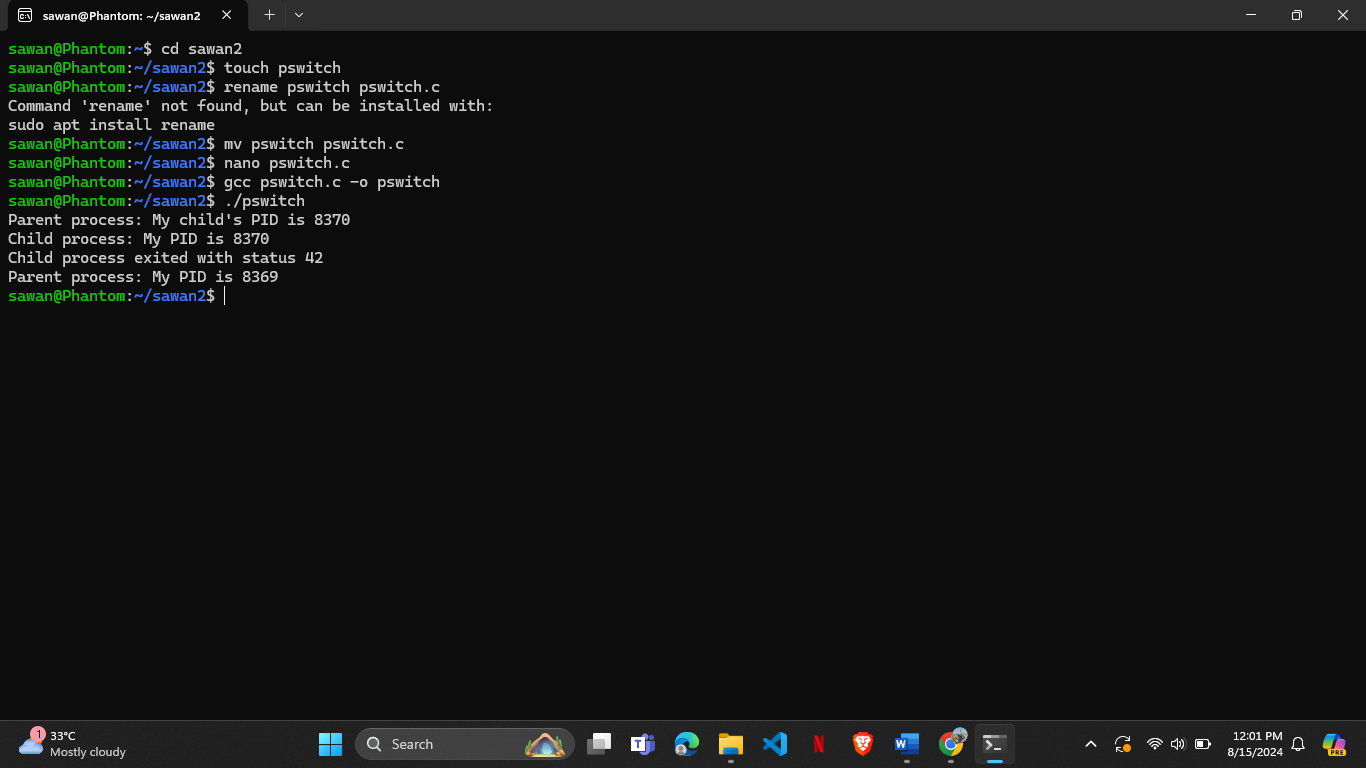
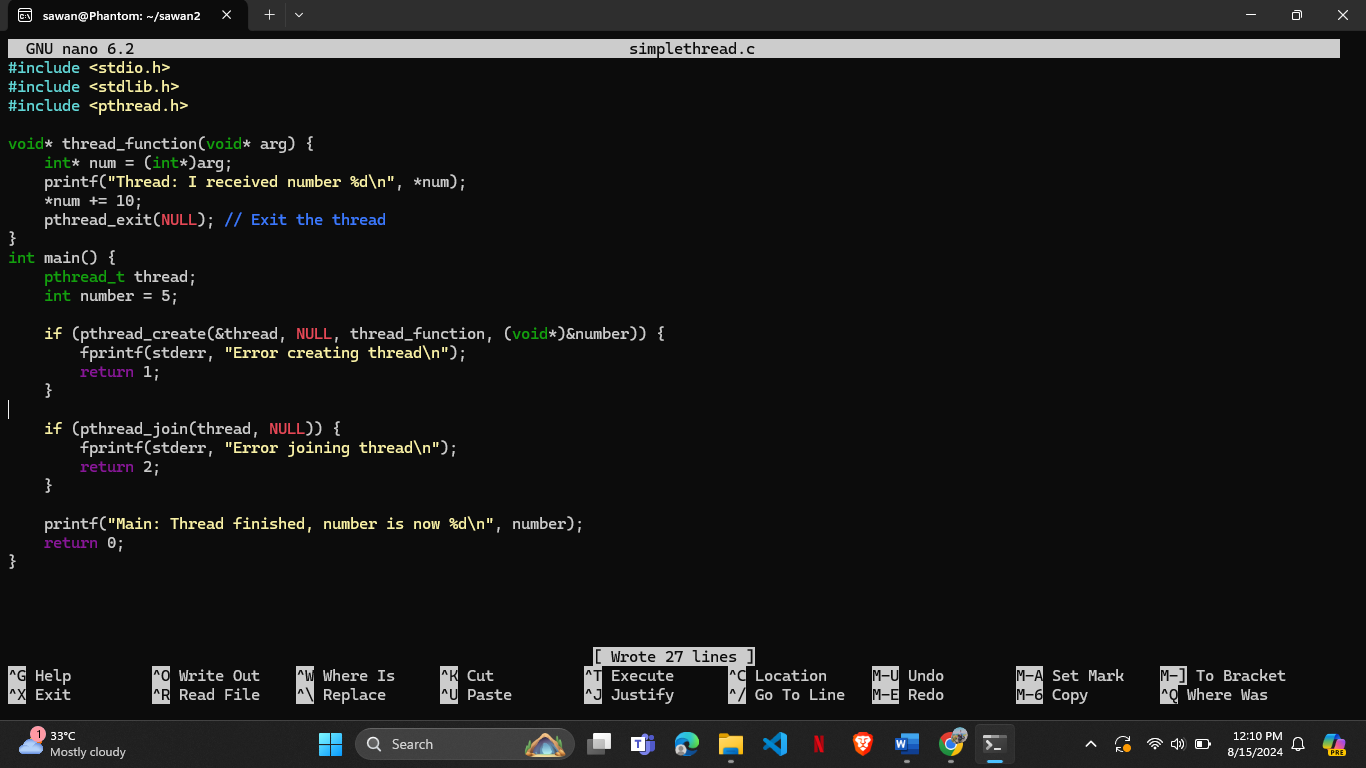
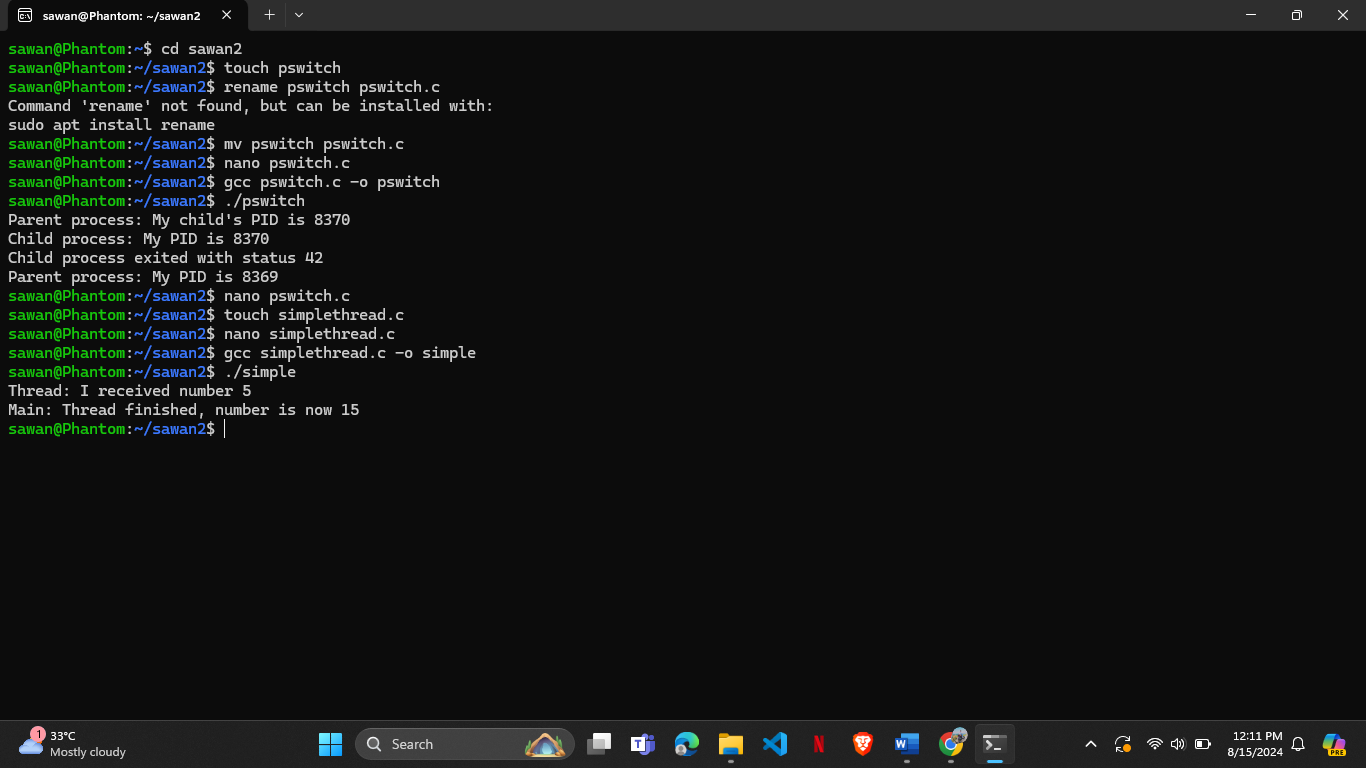
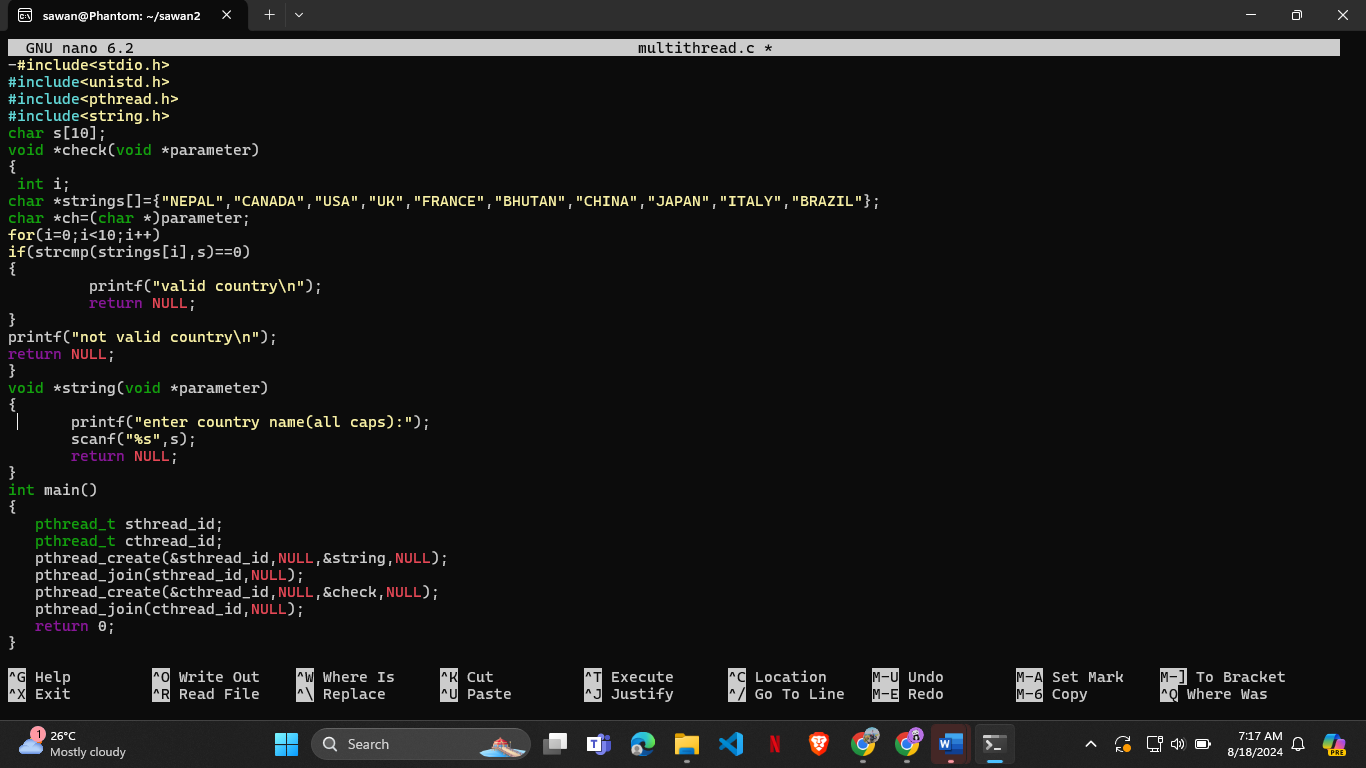
**Output:**

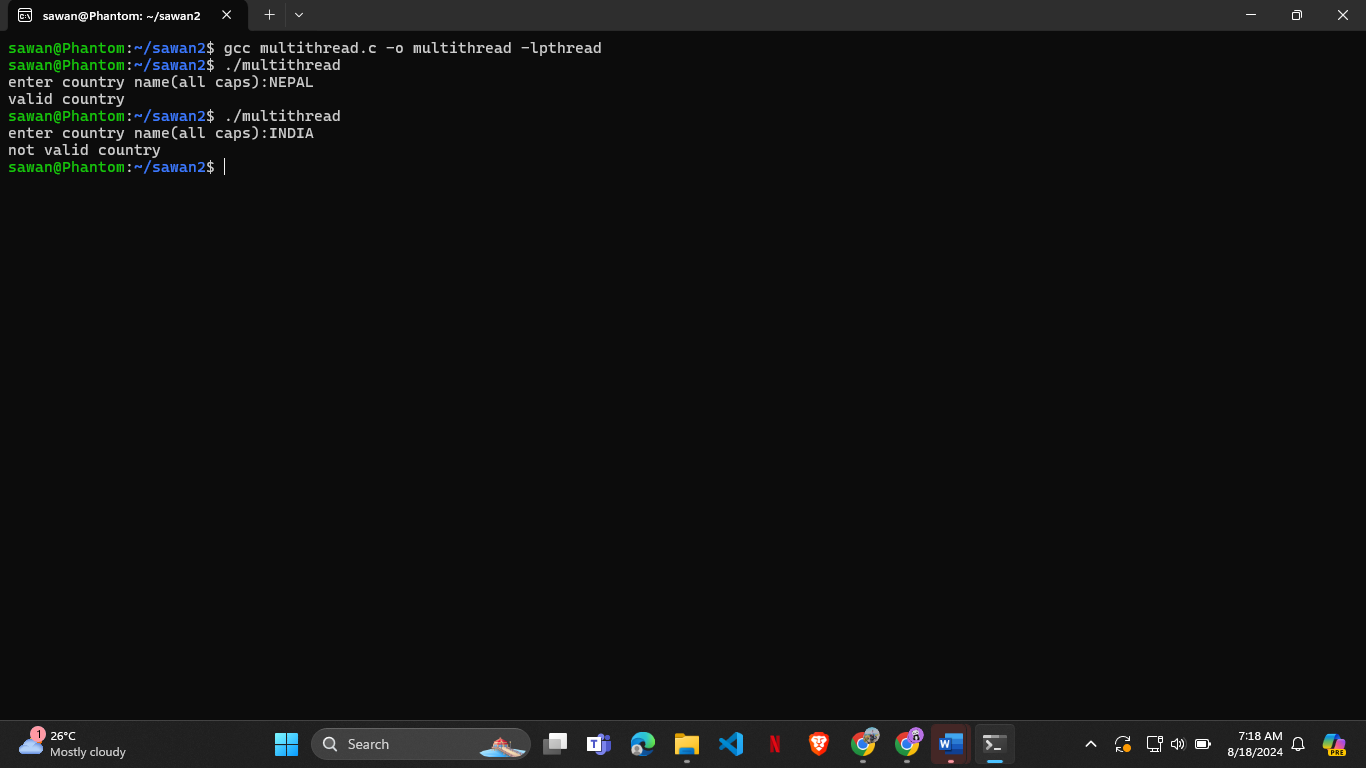
**6. A simple implementation of thread -simplethread.c**

**Output:**

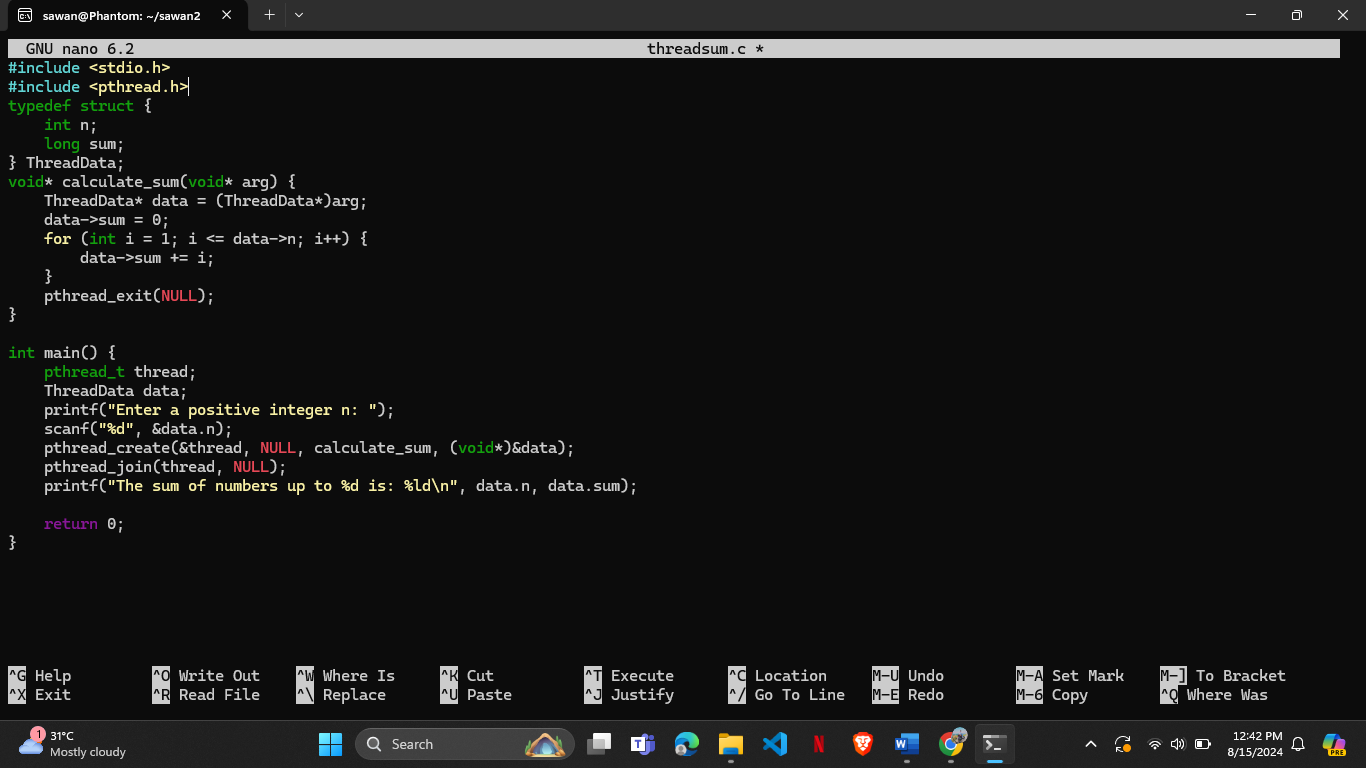
****

**7.Program that have two threads ,one reads a word from keywords and another checks for valid word(you can use your own word list,at least 10, to check validity)- multithread.c**

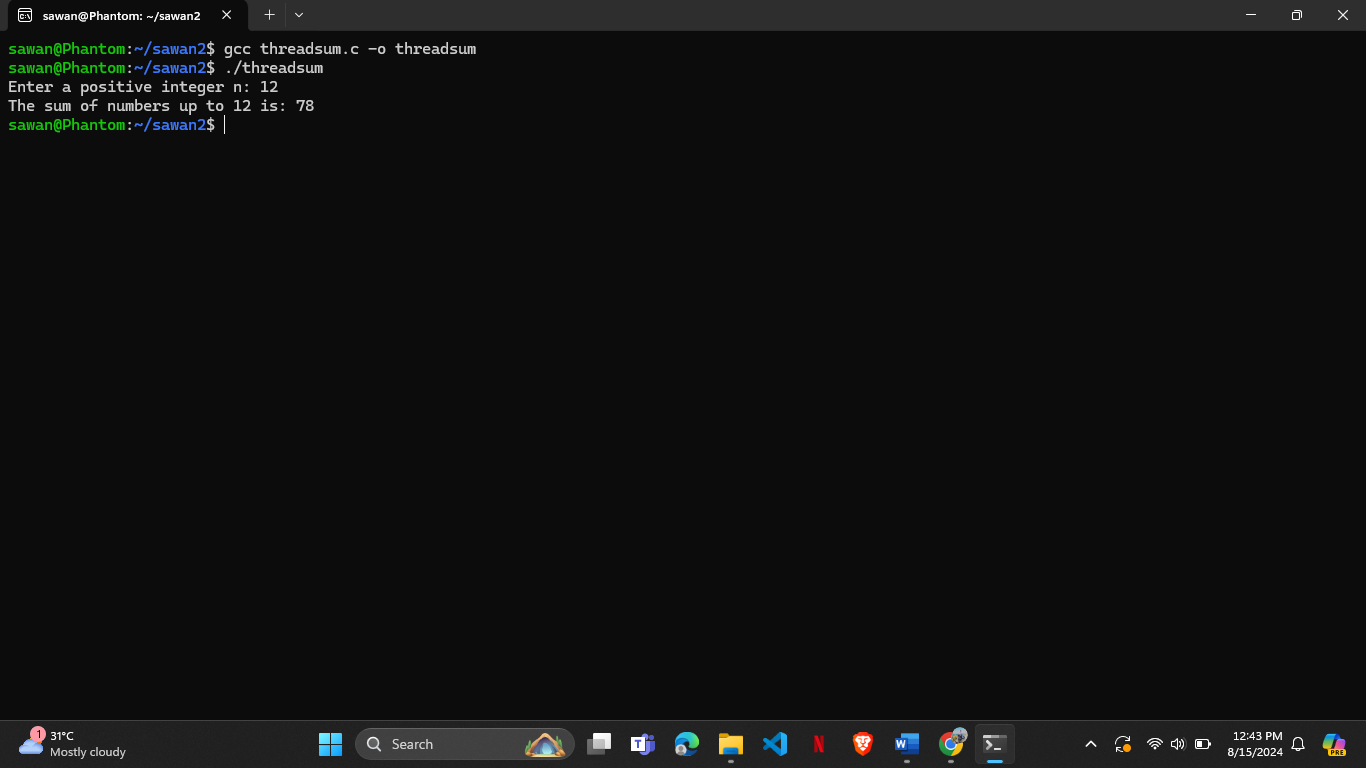
**Output:**



**8.program using thread that prints sum of numbers up to a given position number n- threadsum.c**

****

**Output:**

****

**LAB 4:-Process Scheduling Algorithm**

**1.Simulate First Come First Serve(FCFS) in c.**

#include<stdio.h>

int main()

{

int n,bt[20],wt[20],tat[20],i,j;

float avwt=0,avtat=0;

printf("Enter total number of processes(maximum 20):");

scanf("%d",&n);

printf("\nEnter Process Burst Time\n");

for(i=0;i<n;i++)

{

printf("P[%d]:",i+1);

scanf("%d",&bt[i]);

}

wt[0]=0;

for(i=1;i<n;i++)

{

wt[i]=0;

for(j=0;j<i;j++)

wt[i]+=bt[j];

}

printf("\nProcess \tBurst Time\tWaiting Time\tTurnaround Time");

for(i=0;i<n;i++)

{

tat[i]=bt[i]+wt[i];

avwt+=wt[i];

avtat+=tat[i];

printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);

}

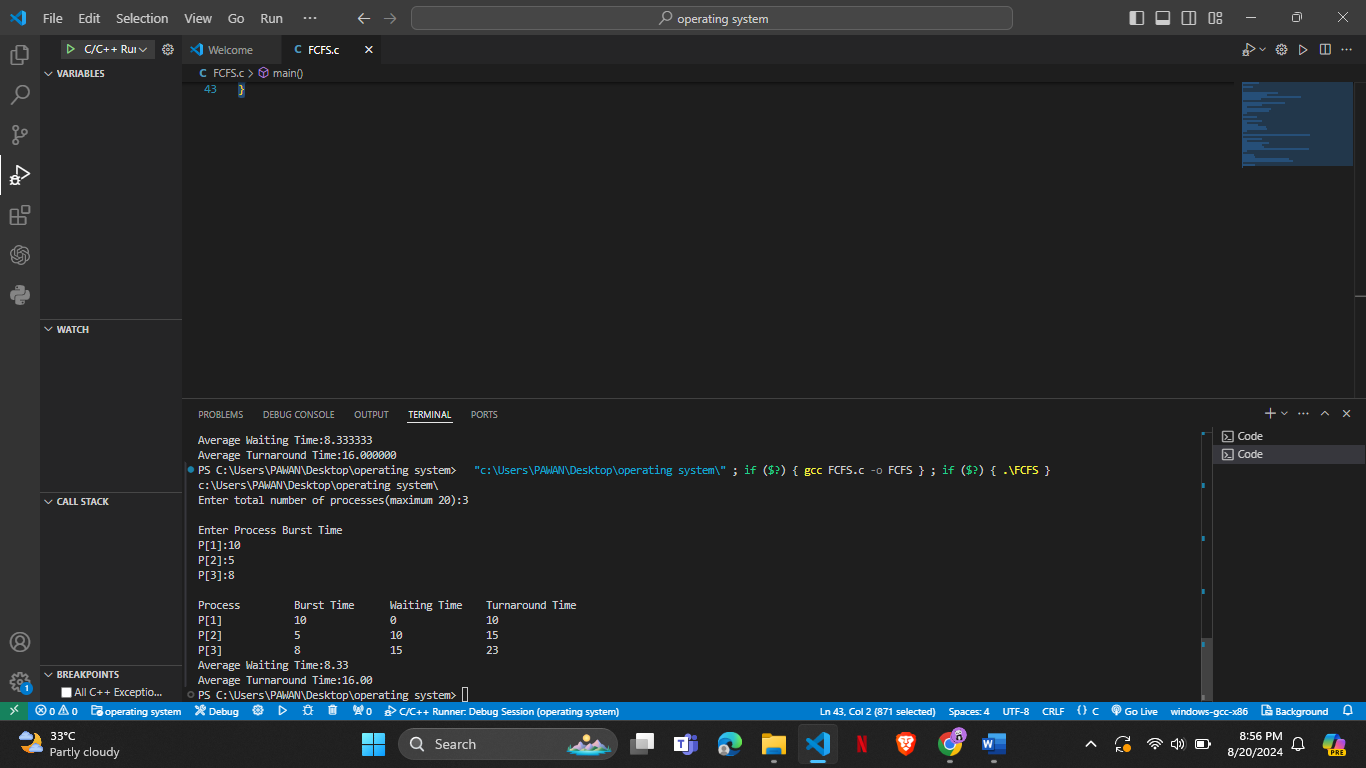
avwt/=i;

avtat/=i;

printf("\nAverage Waiting Time:%.2f",avwt);

printf("\nAverage Turnaround Time:%.2f",avtat);

return 0;}

**OUTPUT:**

**2.Simulate Shortest First Job (SFJ) in c**

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

float avg\_wt,avg\_tat;

printf("Enter number of process:");

scanf("%d",&n);

printf("\nEnter Burst Time:\n");

for(i=0;i<n;i++)

{

printf("p%d:",i+1);

scanf("%d",&bt[i]);

p[i]=i+1;

}

for(i=0;i<n;i++)

{

pos=i;

for(j=i+1;j<n;j++)

{

if(bt[j]<bt[pos])

pos=j;

}

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

for(i=1;i<n;i++)

{

wt[i]=0;

for(j=0;j<i;j++)

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n;

total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");

for(i=0;i<n;i++)

{

tat[i]=bt[i]+wt[i];

total+=tat[i];

printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

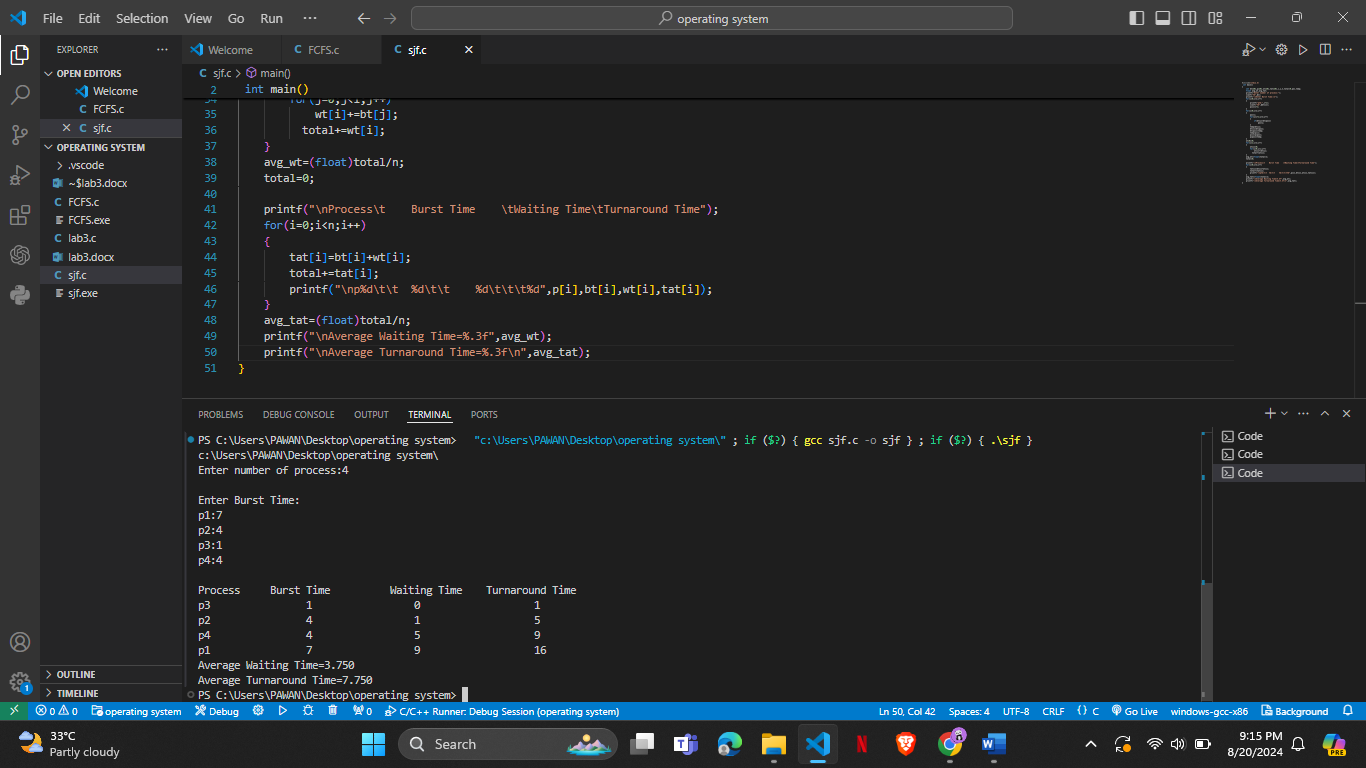
avg\_tat=(float)total/n;

printf("\nAverage Waiting Time=%.3f",avg\_wt);

printf("\nAverage Turnaround Time=%.3f\n",avg\_tat);

}

**OUTPUT:**



**3.Simulate Round Robin (RR) algorithm in c.**

#include<stdio.h>

int main()

{

int i, limit, total = 0, x, counter = 0, time\_quantum;

int wait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10], temp[10];

float average\_wait\_time, average\_turnaround\_time;

printf("\nEnter Total Number of Processes:");

scanf("%d", &limit);

x = limit;

for(i = 0; i < limit; i++)

{

printf("\nEnter Details of Process[%d]\n", i + 1);

printf("Arrival Time:");

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

printf("Burst Time:");

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

temp[i] = burst\_time[i];

}

printf("\nEnter Time Quantum:");

scanf("%d", &time\_quantum);

printf("\nProcess\t\tBurst Time\t Turnaround Time\t Waiting Time");

for(total = 0, i = 0; x != 0;)

{

if(temp[i] <= time\_quantum && temp[i] > 0)

{

total = total + temp[i];

temp[i] = 0;

counter = 1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - time\_quantum;

total = total + time\_quantum;

}

if(temp[i] == 0 && counter == 1)

{

x--;

printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst\_time[i], total - arrival\_time[i], total - arrival\_time[i] - burst\_time[i]);

wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i];

turnaround\_time = turnaround\_time + total - arrival\_time[i];

counter = 0;

}

if(i == limit - 1)

{

i = 0;

}

else if(arrival\_time[i + 1] <= total)

{

i++;

}

else

{

i = 0;

}

}

average\_wait\_time = wait\_time \* 1.0 / limit;

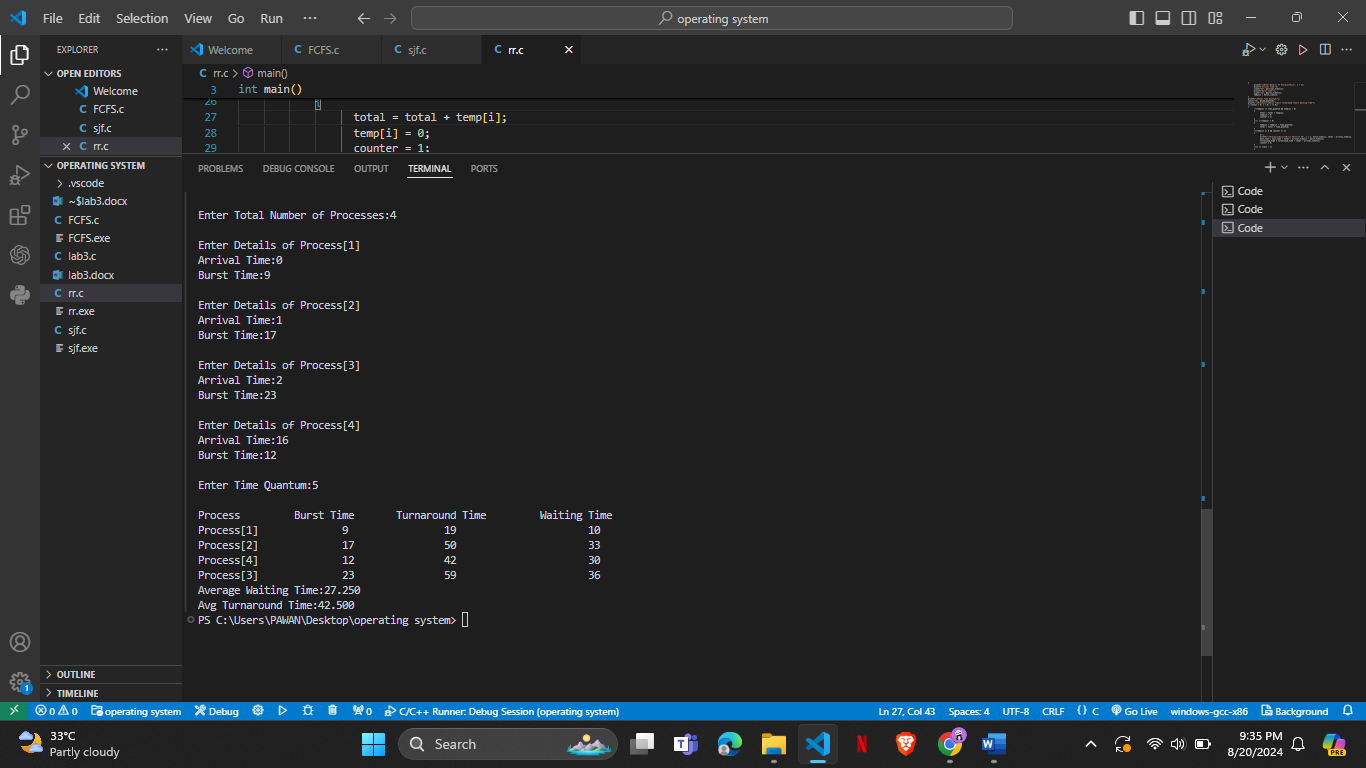
average\_turnaround\_time = turnaround\_time \* 1.0 / limit;

printf("\nAverage Waiting Time:%.3f", average\_wait\_time);

printf("\nAvg Turnaround Time:%.3f\n", average\_turnaround\_time);

return 0;

}

**OUTPUT:**

**4.Simulate Priority Scheluding Algorithm in c.**

#include <stdio.h>

int main()

{

int burstTime[20], process[20], waitingTime[20], turnaroundTime[20], priority[20];

int i, j, numProcesses, totalWaitTime = 0, pos, temp;

float avgWaitTime, avgTurnaroundTime;

printf("Enter Total Number of Processes: ");

scanf("%d", &numProcesses);

printf("\nEnter Burst Time and Priority for each process:\n");

for (i = 0; i < numProcesses; i++) {

printf("Process P[%d]:\n", i + 1);

printf("Burst Time: ");

scanf("%d", &burstTime[i]);

printf("Priority: ");

scanf("%d", &priority[i]);

process[i] = i + 1; // Store process ID

}

for (i = 0; i < numProcesses - 1; i++) {

pos = i;

for (j = i + 1; j < numProcesses; j++) {

if (priority[j] < priority[pos]) {

pos = j;

}

}

temp = priority[i];

priority[i] = priority[pos];

priority[pos] = temp;

temp = burstTime[i];

burstTime[i] = burstTime[pos];

burstTime[pos] = temp;

temp = process[i];

process[i] = process[pos];

process[pos] = temp;

}

waitingTime[0] = 0;

for (i = 1; i < numProcesses; i++) {

waitingTime[i] = 0;

for (j = 0; j < i; j++) {

waitingTime[i] += burstTime[j];

}

totalWaitTime += waitingTime[i];

}

avgWaitTime = (float)totalWaitTime / numProcesses;

totalWaitTime = 0;

printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");

for (i = 0; i < numProcesses; i++) {

turnaroundTime[i] = burstTime[i] + waitingTime[i];

totalWaitTime += turnaroundTime[i];

printf("P[%d]\t%d\t\t%d\t\t%d\n", process[i], burstTime[i], waitingTime[i], turnaroundTime[i]);

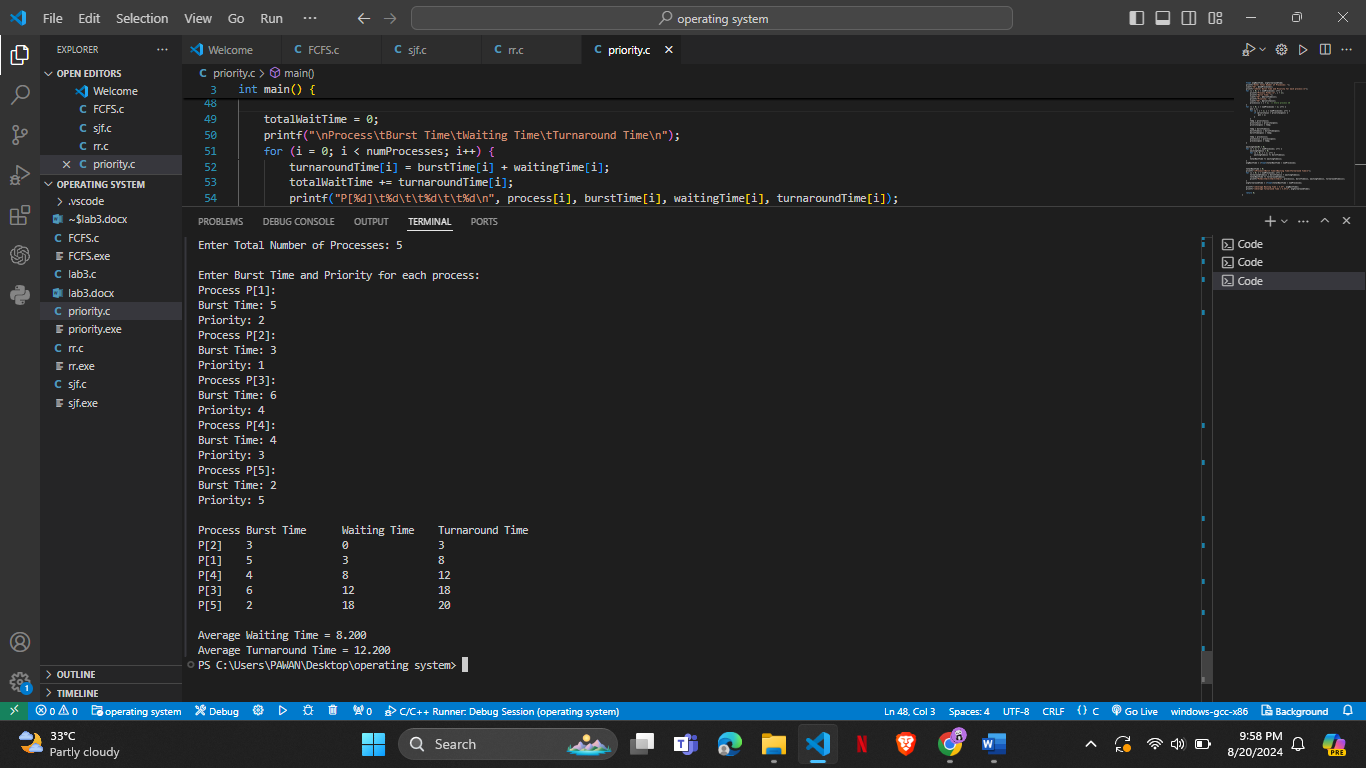
}

avgTurnaroundTime = (float)totalWaitTime / numProcesses;

printf("\nAverage Waiting Time = %.3f", avgWaitTime);

printf("\nAverage Turnaround Time = %.3f\n", avgTurnaroundTime);

return 0;

****}

**OUTPUT:**