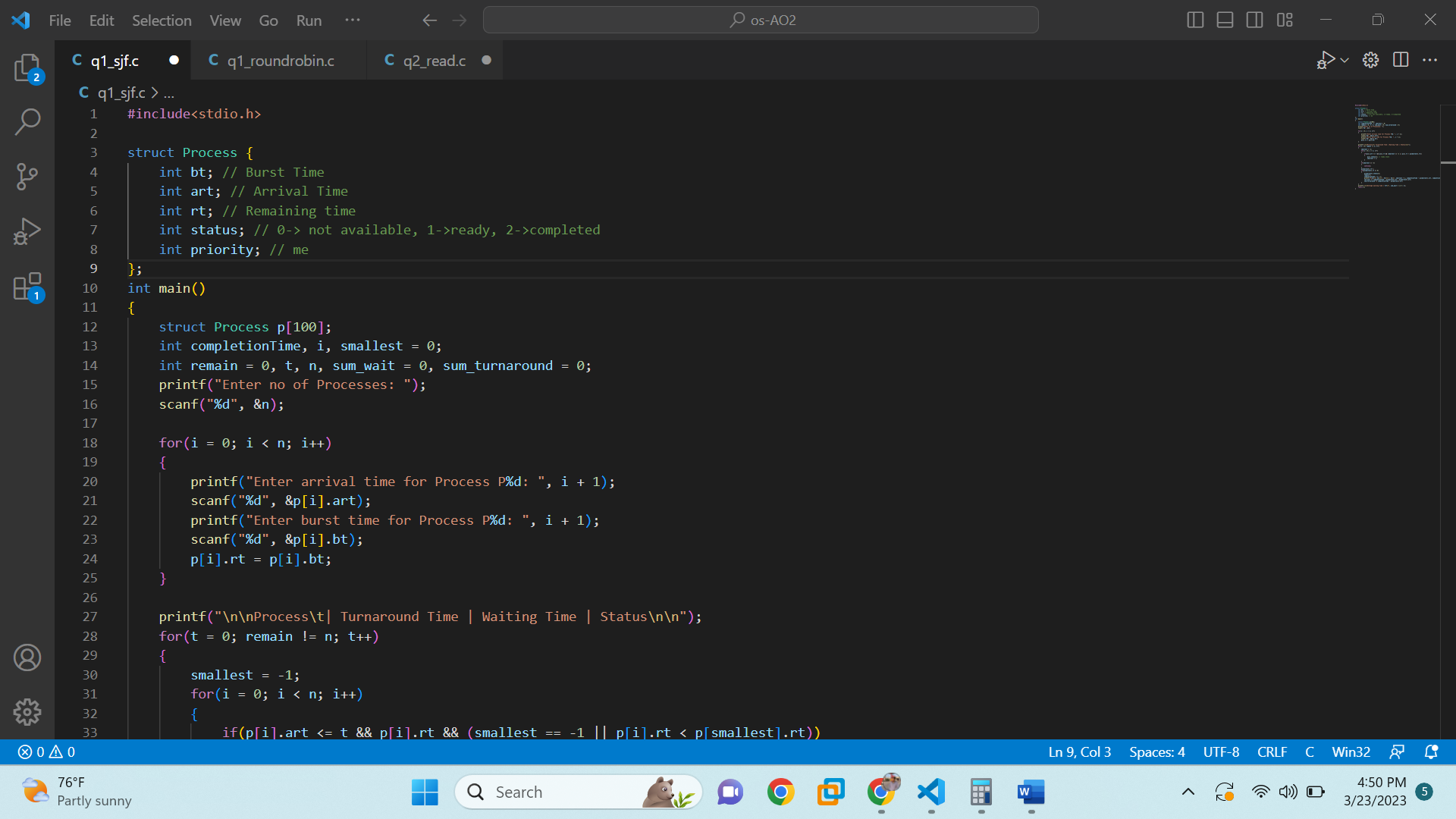
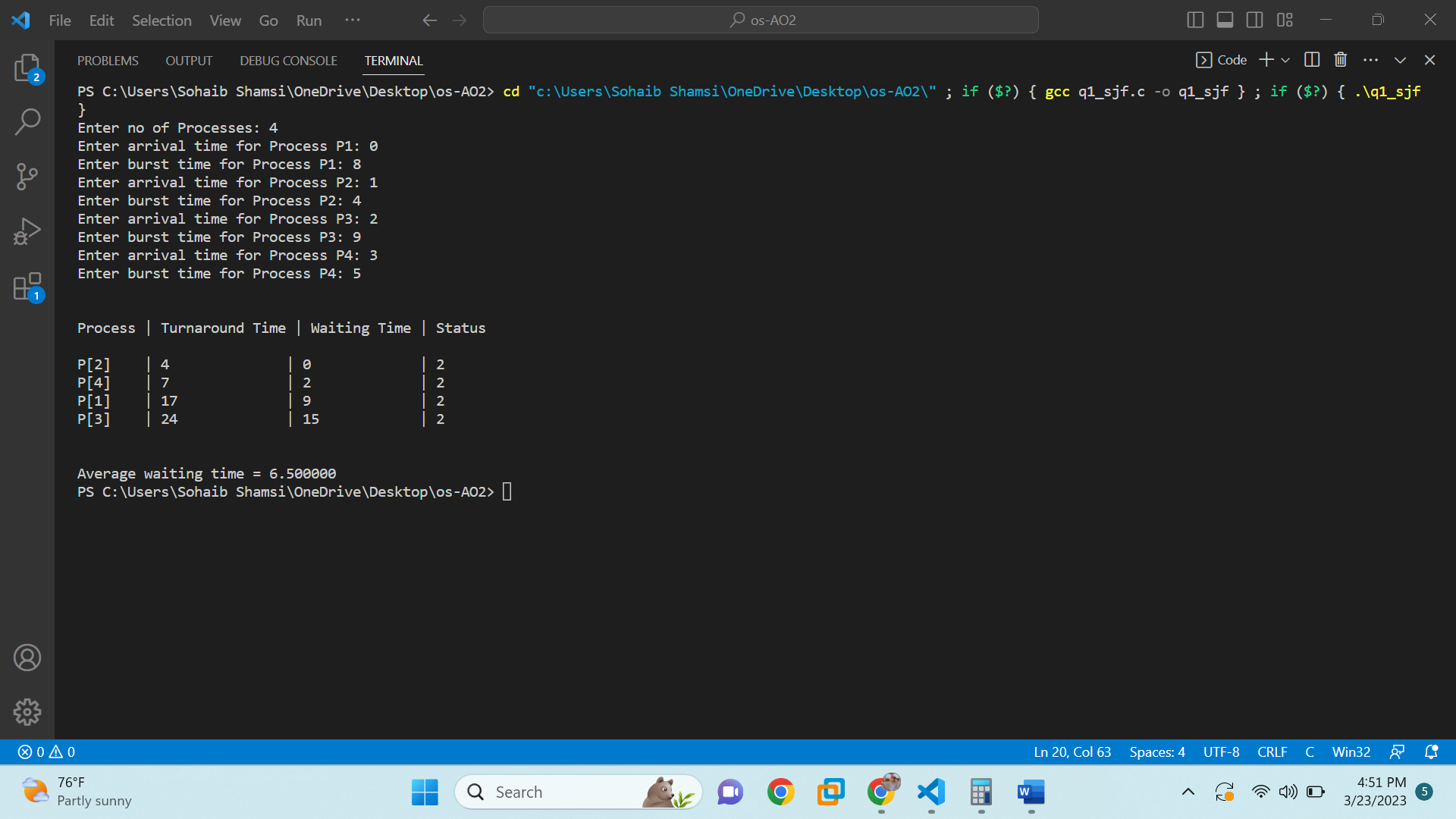
**OS AO2 | 21K-3278-D | Sohaib Sarosh Shamsi**

**Q1 (part a) Shortest Job First (Preempive)**



A screenshot of a computer

Description automatically generated



Graphical user interface, table, Word

Description automatically generated

#include<stdio.h>

struct Process {

int bt; // Burst Time

int art; // Arrival Time

int rt; // Remaining time

int status; // 0-> not available, 1->ready, 2->completed

int priority; // me

};

int main()

{

struct Process p[100];

int completionTime, i, smallest = 0;

int remain = 0, t, n, sum\_wait = 0, sum\_turnaround = 0;

printf("Enter no of Processes: ");

scanf("%d", &n);

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

{

printf("Enter arrival time for Process P%d: ", i + 1);

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

printf("Enter burst time for Process P%d: ", i + 1);

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

p[i].rt = p[i].bt;

}

printf("\n\nProcess\t| Turnaround Time | Waiting Time | Status\n\n");

for(t = 0; remain != n; t++)

{

smallest = -1;

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

{

if(p[i].art <= t && p[i].rt && (smallest == -1 || p[i].rt < p[smallest].rt))

{

p[i].status=1; // ready state

smallest = i;

}

}

if(smallest == -1)

{

continue;

}

p[smallest].rt--;

if(p[smallest].rt == 0)

{

p[smallest].status=2;

remain++;

completionTime = t + 1;

printf("P[%d]\t| %d\t\t | %d\t\t | %d\n", smallest + 1, completionTime - p[smallest].art, completionTime - p[smallest].bt - p[smallest].art, p[smallest].status);

sum\_wait += completionTime - p[smallest].bt - p[smallest].art;

sum\_turnaround += completionTime - p[smallest].art;

}

}

printf("\n\nAverage waiting time = %lf\n", sum\_wait \* 1.0 / n);

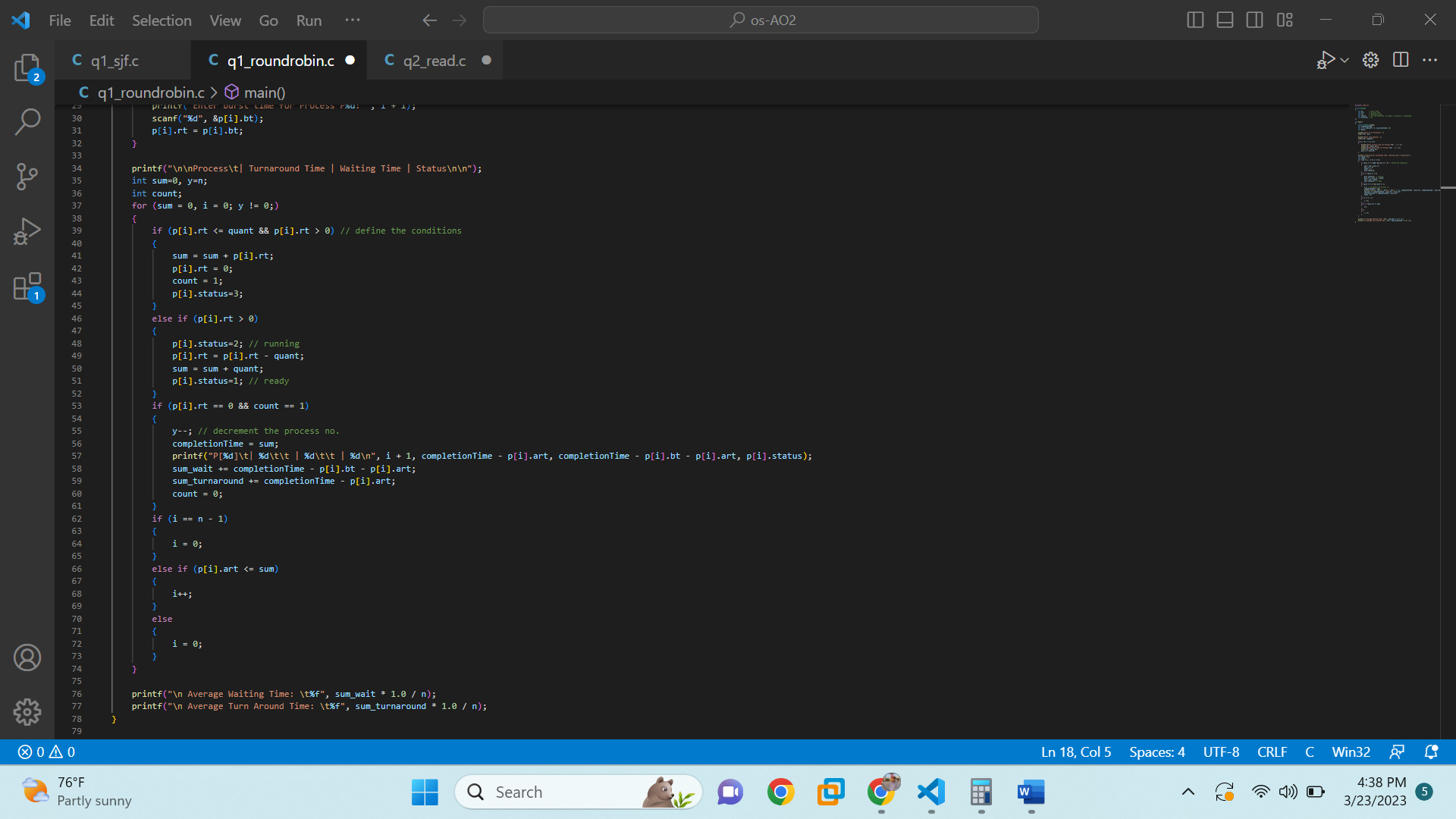
return 0;

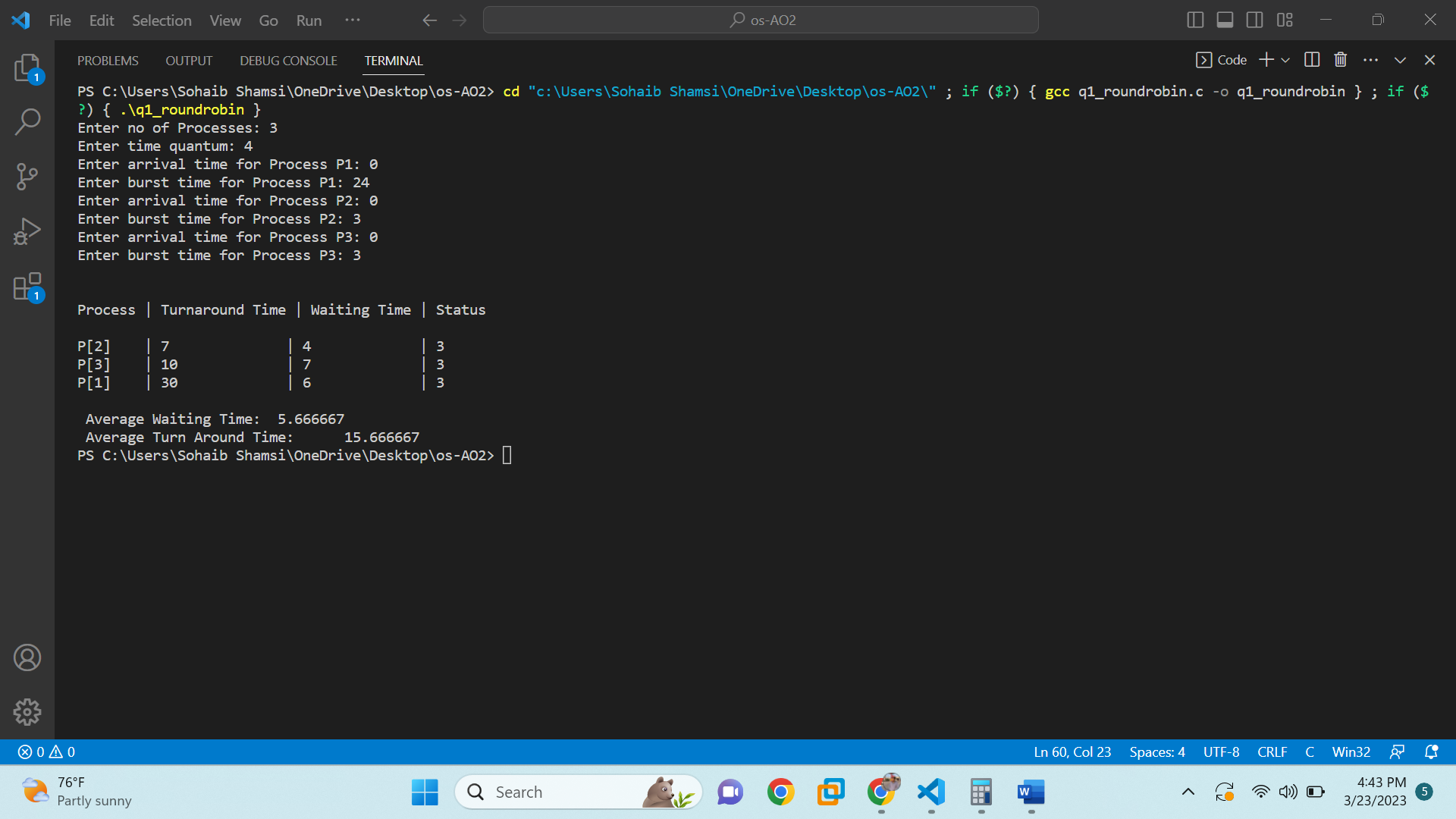
}

**Q1 (part b) Round Robin**

A screenshot of a computer

Description automatically generated





Graphical user interface, application, Word

Description automatically generated

#include <stdio.h>

struct Process

{

int bt; // Burst Time

int art; // Arrival Time

int rt; // Remaining time

int status; // 0-> not available, 1->ready, 2->running, 3->completed

int priority; // me

};

int main()

{

struct Process p[100];

int completionTime;

int i, n, sum\_wait = 0, sum\_turnaround = 0;

int quant;

printf("Enter no of Processes: ");

scanf("%d", &n);

printf("Enter time quantum: ");

scanf("%d", &quant);

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

{

printf("Enter arrival time for Process P%d: ", i + 1);

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

printf("Enter burst time for Process P%d: ", i + 1);

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

p[i].rt = p[i].bt;

}

printf("\n\nProcess\t| Turnaround Time | Waiting Time | Status\n\n");

int sum=0, y=n;

int count;

for (sum = 0, i = 0; y != 0;)

{

if (p[i].rt <= quant && p[i].rt > 0) // define the conditions

{

sum = sum + p[i].rt;

p[i].rt = 0;

count = 1;

p[i].status=3;

}

else if (p[i].rt > 0)

{

p[i].status=2; // running

p[i].rt = p[i].rt - quant;

sum = sum + quant;

p[i].status=1; // ready

}

if (p[i].rt == 0 && count == 1)

{

y--; // decrement the process no.

completionTime = sum;

printf("P[%d]\t| %d\t\t | %d\t\t | %d\n", i + 1, completionTime - p[i].art, completionTime - p[i].bt - p[i].art, p[i].status);

sum\_wait += completionTime - p[i].bt - p[i].art;

sum\_turnaround += completionTime - p[i].art;

count = 0;

}

if (i == n - 1)

{

i = 0;

}

else if (p[i].art <= sum)

{

i++;

}

else

{

i = 0;

}

}

printf("\n Average Waiting Time: \t%f", sum\_wait \* 1.0 / n);

printf("\n Average Turn Around Time: \t%f", sum\_turnaround \* 1.0 / n);

}

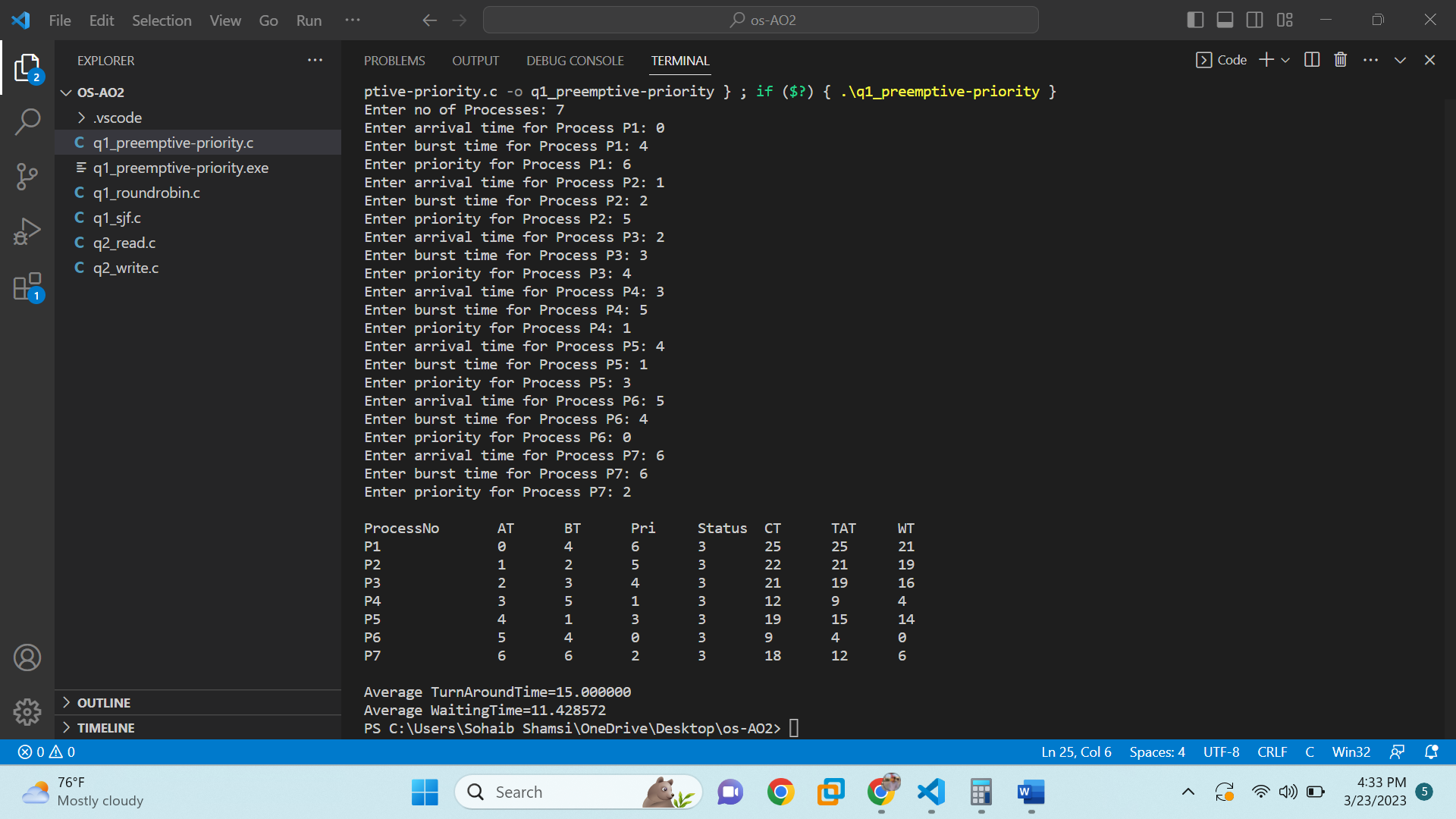
**Q1. (Part c) Priority Scheduling (Preemptive)**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated



Graphical user interface, application, table

Description automatically generated

#include <stdio.h>

#define MIN 99999;

struct Process

{

int bt; // Burst Time

int art; // Arrival Time

int rt; // Remaining time

int status; // 0-> not available, 1->ready, 2->completed

int priority; // me

int ct;

int wt, tat;

int p\_copy;

};

void main()

{

int i, n, c=0, remaining, min\_val, min\_index;

struct Process p[10], temp;

float avgtat = 0, avgwt = 0;

printf("Enter no of Processes: ");

scanf("%d", &n);

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

{

printf("Enter arrival time for Process P%d: ", i + 1);

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

printf("Enter burst time for Process P%d: ", i + 1);

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

printf("Enter priority for Process P%d: ", i + 1);

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

p[i].rt = p[i].bt;

p[i].p\_copy = p[i].priority;

}

remaining = n;

// sort

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

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

if (p[j].art > p[j + 1].art)

{

temp = p[j];

p[j] = p[j + 1];

p[j + 1] = temp;

}

while (remaining > 0)

{

min\_val = p[0].priority, min\_index = 0;

for (int j = 0; j < n && p[j].art <= c; j++)

if (p[j].priority < min\_val)

min\_val = p[j].priority, min\_index = j;

i = min\_index;

p[i].ct = c = c + 1;

p[i].rt--;

if (p[i].rt == 0)

{

p[i].status=3; // completed

p[i].priority = MIN;

remaining--;

}

}

printf("\nProcessNo\tAT\tBT\tPri\tStatus\tCT\tTAT\tWT\n");

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

{

p[i].tat = p[i].ct - p[i].art;

avgtat += p[i].tat;

p[i].wt = p[i].tat - p[i].bt;

avgwt += p[i].wt;

printf("P%d\t\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n", i+1, p[i].art, p[i].bt, p[i].p\_copy, p[i].status,p[i].ct, p[i].tat, p[i].wt);

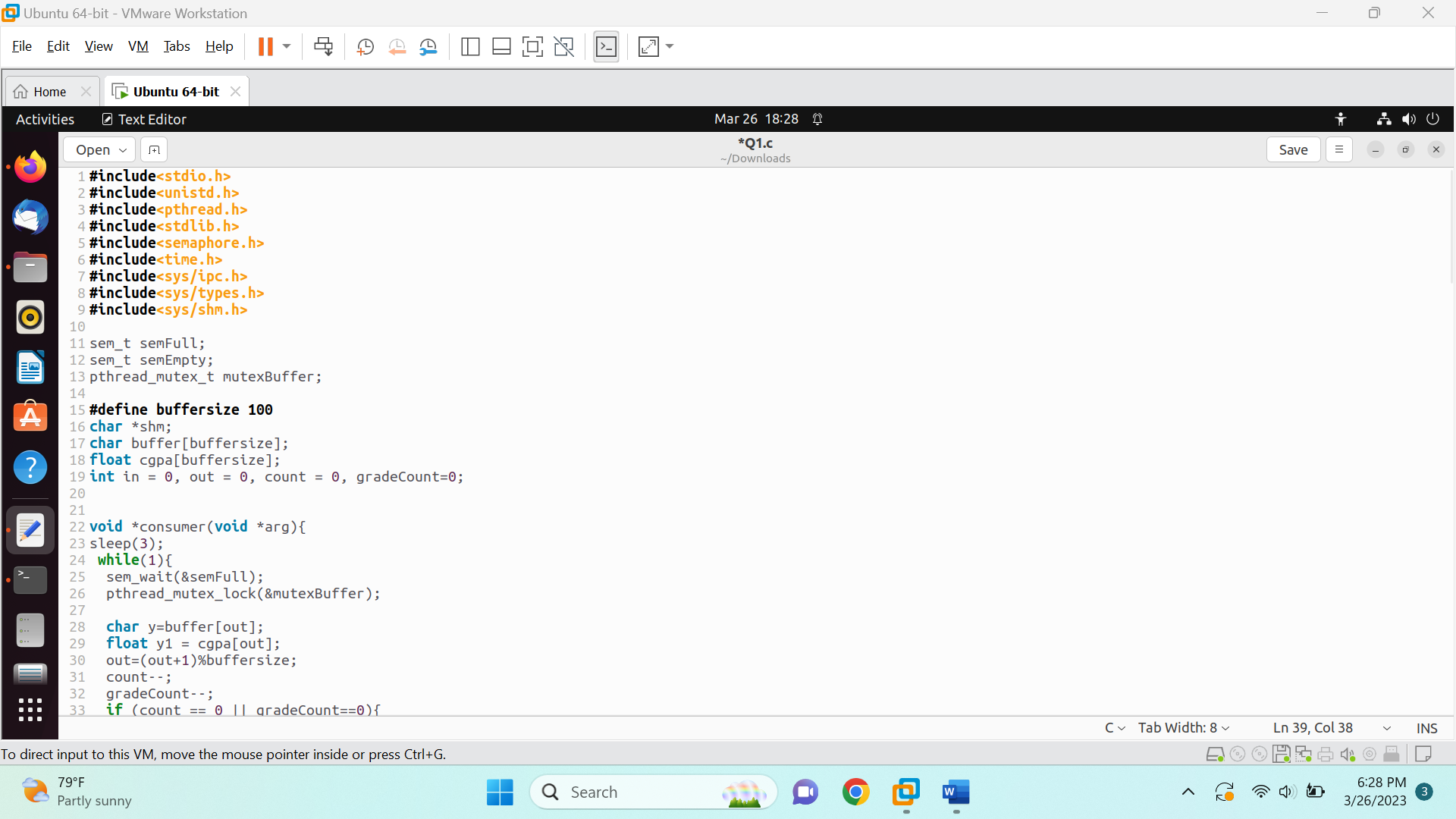
}

avgtat /= n, avgwt /= n;

printf("\nAverage TurnAroundTime=%f\nAverage WaitingTime=%f", avgtat, avgwt);

}

**QUESTION 2**

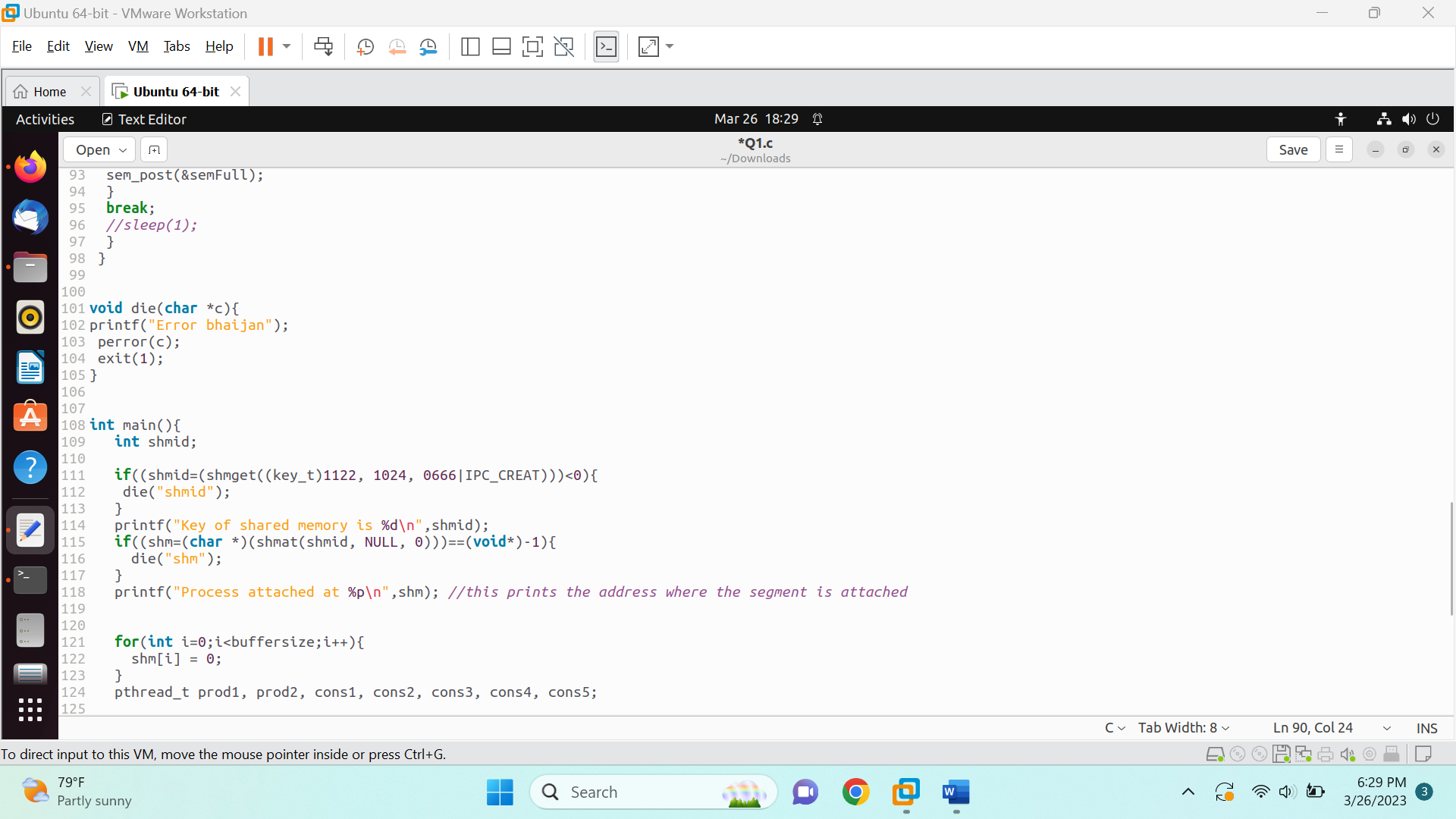


Graphical user interface, text, application

Description automatically generated

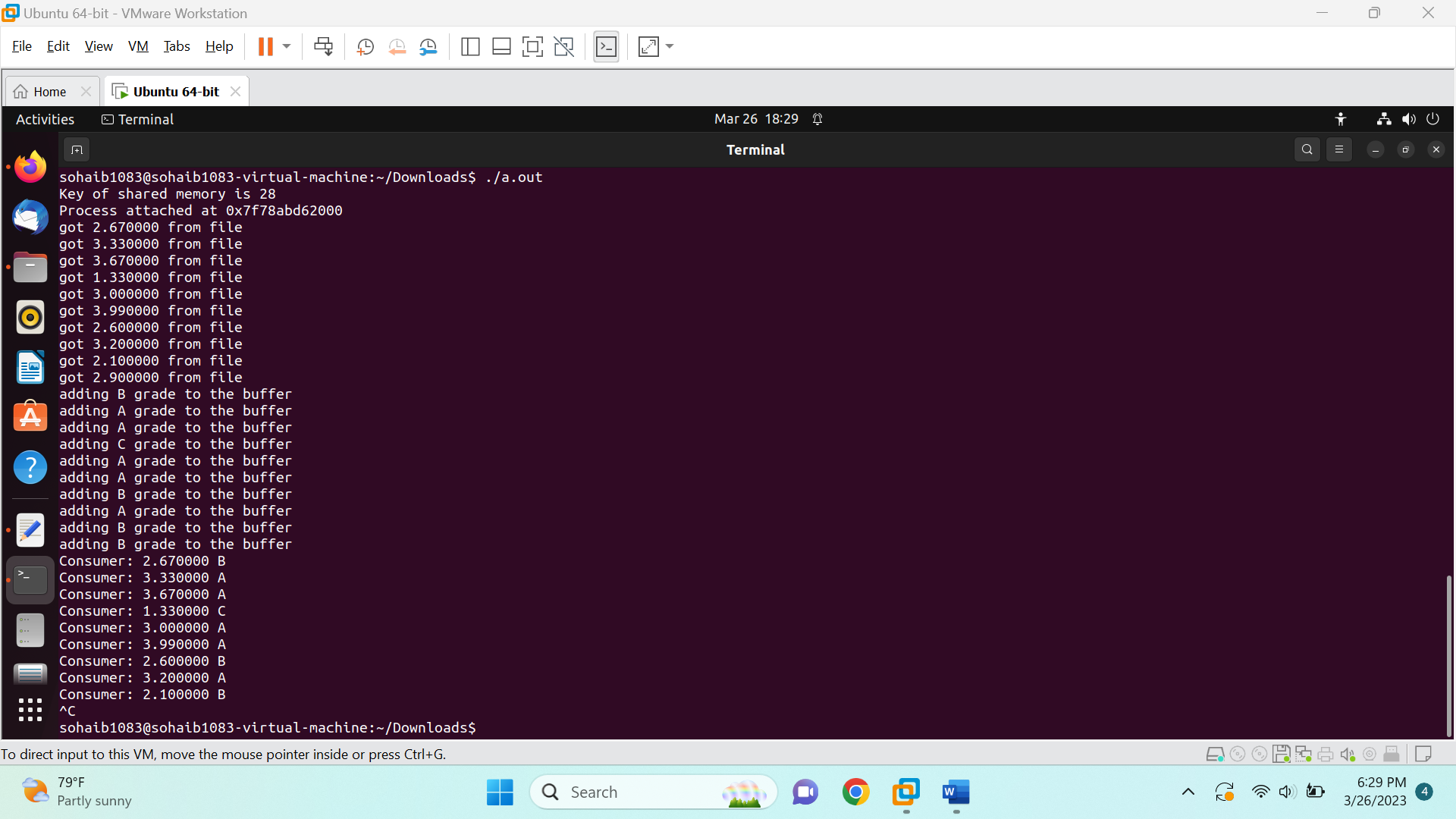
Graphical user interface, text, application

Description automatically generated



Graphical user interface, text, application

Description automatically generated



#include<stdio.h>

#include<unistd.h>

#include<pthread.h>

#include<stdlib.h>

#include<semaphore.h>

#include<time.h>

#include<sys/ipc.h>

#include<sys/types.h>

#include<sys/shm.h>

sem\_t semFull;

sem\_t semEmpty;

pthread\_mutex\_t mutexBuffer;

#define buffersize 100

char \*shm;

char buffer[buffersize];

float cgpa[buffersize];

int in = 0, out = 0, count = 0, gradeCount=0;

void \*consumer(void \*arg){

sleep(3);

while(1){

sem\_wait(&semFull);

pthread\_mutex\_lock(&mutexBuffer);

char y=buffer[out];

float y1 = cgpa[out];

out=(out+1)%buffersize;

count--;

gradeCount--;

if (count == 0 || gradeCount==0){

break;

}

printf("Consumer: %f %c\n", y1, y);

sem\_post(&semEmpty);

pthread\_mutex\_unlock(&mutexBuffer);

}

}

void \*producer1(void \*arg){

float x;

FILE \*fp;

fp=fopen("text.txt", "r");

for(int i=0;i<10;i++){

sem\_wait(&semEmpty);

pthread\_mutex\_lock(&mutexBuffer);

fscanf(fp,"%f", &x);

printf("got %f from file\n", x);

cgpa[in]=x;

in=(in+1)%buffersize;

count++;

sem\_post(&semFull);

pthread\_mutex\_unlock(&mutexBuffer);

}

fclose(fp);

}

void \*producer2(void \*arg){

sleep(1);

in=0;

while (1)

{

sem\_wait(&semEmpty);

pthread\_mutex\_lock(&mutexBuffer);

for (int i=0; i<10; i++){

if (cgpa[in] >= 3){

buffer[in] = 'A';

printf("adding A grade to the buffer\n");

}

else if (cgpa[in] >= 2 && cgpa[in] < 3){

buffer[in] = 'B';

printf("adding B grade to the buffer\n");

}

else if (cgpa[in] >= 1 && cgpa[in] < 2){

buffer[in] = 'C';

printf("adding C grade to the buffer\n");

}

else{

buffer[in] = 'D';

printf("adding D grade to the buffer\n");

}

in=(in+1)%buffersize;

gradeCount++;

pthread\_mutex\_unlock(&mutexBuffer);

sem\_post(&semFull);

}

break;

//sleep(1);

}

}

void die(char \*c){

printf("Error bhaijan");

perror(c);

exit(1);

}

int main(){

int shmid;

if((shmid=(shmget((key\_t)1122, 1024, 0666|IPC\_CREAT)))<0){

die("shmid");

}

printf("Key of shared memory is %d\n",shmid);

if((shm=(char \*)(shmat(shmid, NULL, 0)))==(void\*)-1){

die("shm");

}

printf("Process attached at %p\n",shm); //this prints the address where the segment is attached

for(int i=0;i<buffersize;i++){

shm[i] = 0;

}

pthread\_t prod1, prod2, cons1, cons2, cons3, cons4, cons5;

sem\_init(&semFull, 0, 0);

sem\_init(&semEmpty, 0, buffersize);

pthread\_mutex\_init(&mutexBuffer, NULL);

pthread\_create(&prod1, NULL, producer1, NULL);

pthread\_create(&prod2, NULL, producer2, NULL);

// 5 threads for consumer

pthread\_create(&cons1, NULL, consumer, NULL);

pthread\_create(&cons2, NULL, consumer, NULL);

pthread\_create(&cons3, NULL, consumer, NULL);

pthread\_create(&cons4, NULL, consumer, NULL);

pthread\_create(&cons5, NULL, consumer, NULL);

pthread\_join(prod1, NULL);

pthread\_join(prod2, NULL);

pthread\_join(cons1, NULL);

pthread\_join(cons2, NULL);

pthread\_join(cons3, NULL);

pthread\_join(cons4, NULL);

pthread\_join(cons5, NULL);

sem\_destroy(&semFull);

sem\_destroy(&semEmpty);

pthread\_mutex\_destroy(&mutexBuffer);

return 0;

}