

# ***CSA0496-OPERATING SYSTEM WITH TASK MIGRATION***

***DAY-01***

***NAME:B.VINEETHA***

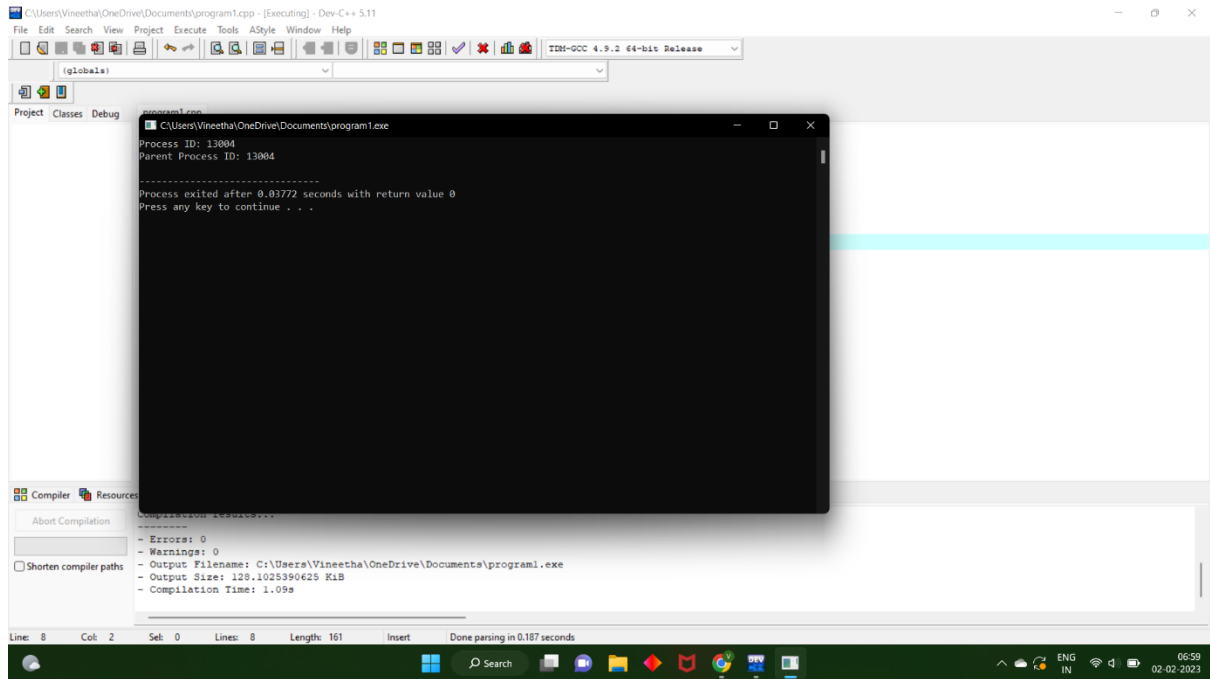
***REG.NO:192110487***

***1. Create a new process by invoking the appropriate system call. Get the process identifier of the currently running process and its respective parent using system calls and display the same using a C program.***

***PROGRAM:***

```
#include<stdio.h>
#include<unistd.h>
int main()
{
    printf("Process ID: %d\n", getpid() );
    printf("Parent Process ID: %d\n", getpid() );
    return 0;
}
```

***OUTPUT:***



**2. Identify the system calls to copy the content of one file to another and illustrate the same using a C program.**

**PROGRAM:**

```
#include<stdio.h>

#include<stdlib.h>

int main()
{
    FILE *f1,*f2;
    char filename[100],c;
    f1=fopen("D:\\DEV++\\test2.c","r");
    f2=fopen("D:\\DEV++\\test1.txt","w");
    c=fgetc(f1);
    while(c!=EOF)
    {
        fputc(c,f2);
        c=fgetc(f1);
    }
    printf("CONTENTS COPIED SUCCESSFULLY....");
    fclose(f1);
}
```

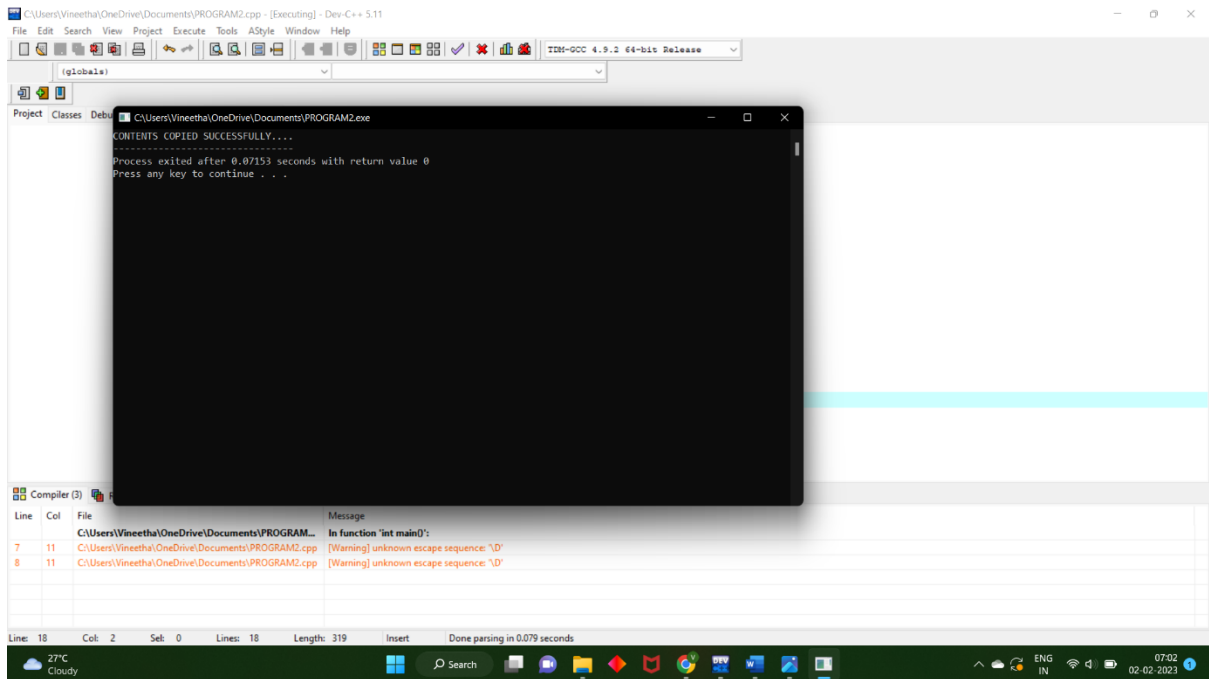
```

fclose(f2);

}

```

### OUTPUT:



**3. Design a CPU scheduling program with C using First Come First Served technique with the following considerations.**

**a. All processes are activated at time 0.**

**b. Assume that no process waits on I/O devices.**

### PROGRAM:

```

#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);
    }
}

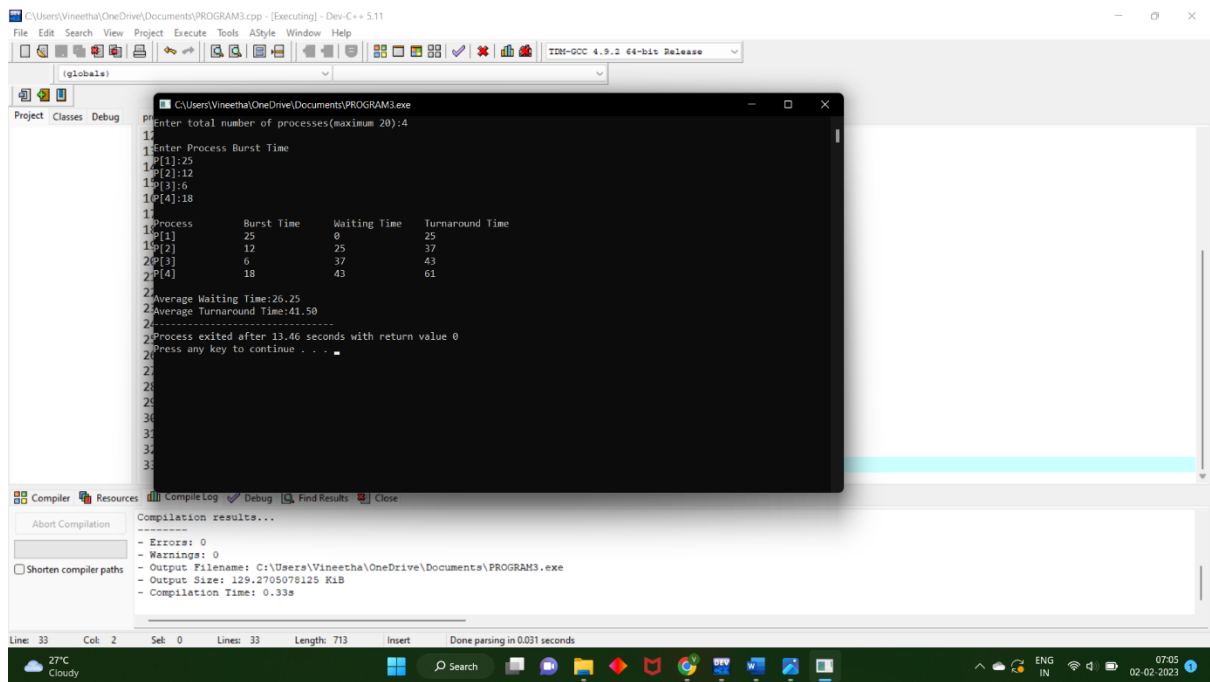
```

```

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\t\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("\n\nAverage Waiting Time:%.2f",avwt);
printf("\n\nAverage Turnaround Time:%.2f",avtat);
return 0;
}

```

**OUTPUT:**



**4. Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next.**

**PROGRAM:**

```
#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\t tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];
    total+=tat[i];

    printf("\np%d\t %d\t\t %d\t\t%d",p[i],bt[i],wt[i],tat[i]);
}
avg_tat=(float)total/n;

```

```

printf("\nAverage Waiting Time=%f",avg_wt);
printf("\nAverage Turnaround Time=%f",avg_tat);
}

```

### OUTPUT:

```

C:\Users\vajan\OneDrive\Pictures\Documents\PROGRAM 4 OS.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug PROGRAM 1 OS.cpp PROGRAM 2 OS.cpp PROGRAM 3 OS.cpp PROGRAM 4 OS.cpp
27 temp=p[i];
C:\Users\vajan\OneDrive\Pictures\Documents\PROGRAM 4 OS.exe
Enter number of process:3
nEnter Burst Time:np1:2
p2:3
p3:3
nProcesst Burst Time tWaiting TimeTurnaround Timep1tt 2tt 0ttt2np2tt 3tt 2ttt5np3tt 3tt 5ttt8nnAvere
ge Waiting Times=2.333333nAverage Turnaround Time=5.000000n
Process exited after 7.363 seconds with return value 0
Press any key to continue . . .

```

**5. Construct a scheduling program with C that selects the waiting process with the highest priority to execute next.**

### PROGRAM:

```

#include<stdio.h>

struct priority_scheduling {
    char process_name;
    int burst_time;
    int waiting_time;
    int turn_around_time;
    int priority;
};

int main() {
    int number_of_process;
    int total = 0;

```

```

struct priority_scheduling temp_process;
int ASCII_number = 65;
int position;
float average_waiting_time;
float average_turnaround_time;
printf("Enter the total number of Processes: ");
scanf("%d", & number_of_process);
struct priority_scheduling process[number_of_process];
printf("\nPlease Enter the Burst Time and Priority of each process:\n");
for (int i = 0; i < number_of_process; i++) {
    process[i].process_name = (char) ASCII_number;
    printf("\nEnter the details of the process %c \n", process[i].process_name);
    printf("Enter the burst time: ");
    scanf("%d", & process[i].burst_time);
    printf("Enter the priority: ");
    scanf("%d", & process[i].priority);
    ASCII_number++;
}
for (int i = 0; i < number_of_process; i++) {
    position = i;
    for (int j = i + 1; j < number_of_process; j++) {
        if (process[j].priority > process[position].priority)
            position = j;
    }
    temp_process = process[i];
    process[i] = process[position];
    process[position] = temp_process;
}
process[0].waiting_time = 0;
for (int i = 1; i < number_of_process; i++) {
    process[i].waiting_time = 0;
    for (int j = 0; j < i; j++) {
        process[i].waiting_time += process[j].burst_time;
    }
}

```



```

    }

    total += process[i].waiting_time;
}

average_waiting_time = (float) total / (float) number_of_process;

total = 0;

printf("\n\nProcess_name \t Burst Time \t Waiting Time \t Turnaround Time\n");
printf("-----\n");

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

    process[i].turn_around_time = process[i].burst_time + process[i].waiting_time;
    total += process[i].turn_around_time;

    printf("\t %c \t %d \t %d \t %d", process[i].process_name, process[i].burst_time,
        process[i].waiting_time, process[i].turn_around_time);

    printf("\n-----\n");
}

average_turnaround_time = (float) total / (float) number_of_process;

printf("\n\n Average Waiting Time : %f", average_waiting_time);

printf("\n\n Average Turnaround Time: %f\n", average_turnaround_time);

return 0;
}

```

### OUTPUT:

The screenshot shows the execution of a C++ program in Dev-C++. The program prompts the user to enter the total number of processes (3) and then the burst time and priority for each process. The output displays a table of process details and the calculated average waiting and turnaround times.

Process_name	Burst Time	Waiting Time	Turnaround Time
B	4	0	4
A	3	4	7
C	3	7	10

Average Waiting Time : 3.66667  
Average Turnaround Time: 7.00000

**6. Construct a C program to implement pre-emptive priority scheduling algorithm.**

**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i, NOP, sum=0, count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
    float avg_wt, avg_tat;
    printf(" Total number of process in the system: ");
    scanf("%d", &NOP);
    y = NOP;
    for(i=0; i<NOP; i++)
    {
        printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
        printf(" Arrival time is: \t");
        scanf("%d", &at[i]);
        printf(" \nBurst time is: \t");
        scanf("%d", &bt[i]);
        temp[i] = bt[i];
    }
    printf("Enter the Time Quantum for the process: \t");
    scanf("%d", &quant);
    printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
    for(sum=0, i = 0; y!=0; )
    {
        if(temp[i] <= quant && temp[i] > 0)
        {
            sum = sum + temp[i];
            temp[i] = 0;
            count=1;
        }
    }
```

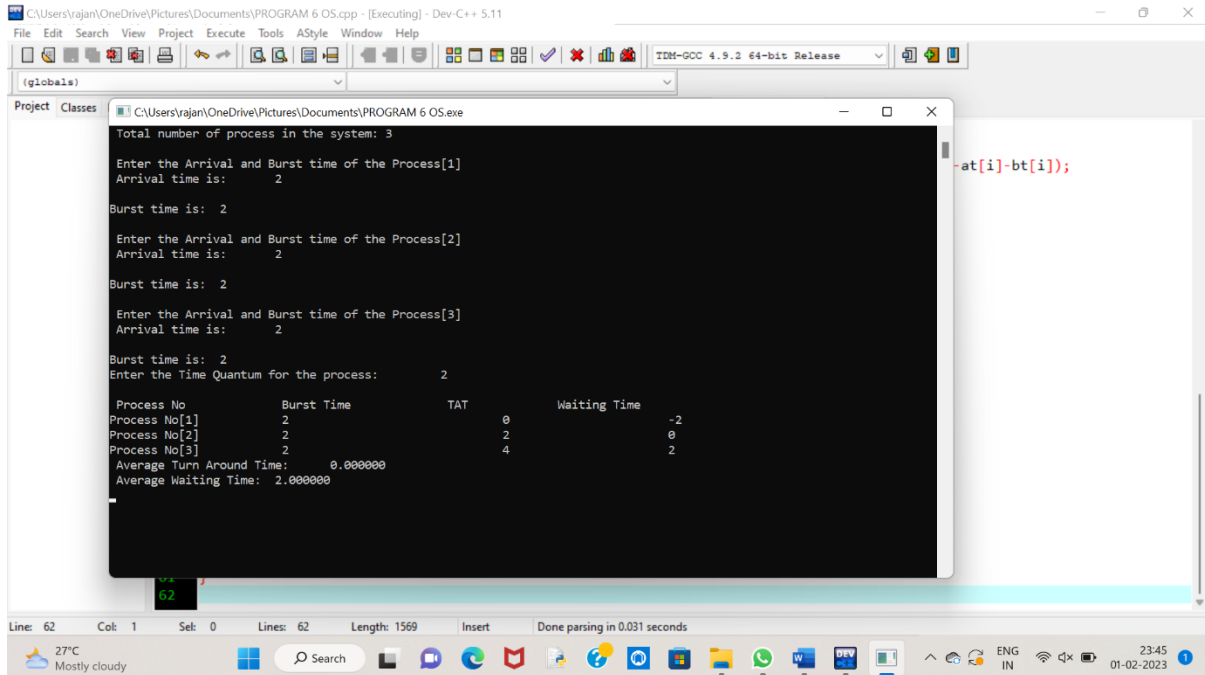
```

else if(temp[i] > 0)
{
    temp[i] = temp[i] - quant;
    sum = sum + quant;
}
if(temp[i]==0 && count==1)
{
    y--;
    printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-
bt[i]);
    wt = wt+sum-at[i]-bt[i];
    tat = tat+sum-at[i];
    count =0;
}
if(i==NOP-1)
{
    i=0;
}
else if(at[i+1]<=sum)
{
    i++;
}
else
{
    i=0;
}
}
avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
getch();

```

}

## OUTPUT:



```
C:\Users\rajan\OneDrive\Pictures\Documents\PROGRAM 6 OS.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes C:\Users\rajan\OneDrive\Pictures\Documents\PROGRAM 6 OS.exe
Total number of process in the system: 3
Enter the Arrival and Burst time of the Process[1]
Arrival time is: 2
Burst time is: 2
Enter the Arrival and Burst time of the Process[2]
Arrival time is: 2
Burst time is: 2
Enter the Arrival and Burst time of the Process[3]
Arrival time is: 2
Burst time is: 2
Enter the Time Quantum for the process: 2
Process No      Burst Time      TAT      Waiting Time
Process No[1]   2                0        -2
Process No[2]   2                2        0
Process No[3]   2                4        2
Average Turn Around Time: 0.000000
Average Waiting Time: 2.000000
```

## 7. Construct a C program to implement non-preemptive SJF algorithm.

### PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>

int main()
{
    int i;
    void *shared_memory;
    char buff[100];
    int shmid;
    shmid=shmget((key_t)2345, 1024, 0666|IPC_CREAT);
    printf("Key of shared memory is %d\n",shmid);
    shared_memory=shmat(shmid,NULL,0);
    printf("Process attached at %p\n",shared_memory);
```

```

printf("Enter some data to write to shared memory\n");
read(0,buff,100);
strcpy(shared_memory,buff);
printf("You wrote : %s\n",(char *)shared_memory);
}

```

### OUTPUT:

```

C:\Users\vajan\OneDrive\Pictures\Documents\PROGRAM 6 OS.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes
C:\Users\vajan\OneDrive\Pictures\Documents\PROGRAM 6 OS.exe
Total number of process in the system: 3
Enter the Arrival and Burst time of the Process[1]
Arrival time is: 2
Burst time is: 2
Enter the Arrival and Burst time of the Process[2]
Arrival time is: 2
Burst time is: 2
Enter the Arrival and Burst time of the Process[3]
Arrival time is: 2
Burst time is: 2
Enter the Time Quantum for the process: 2
Process No      Burst Time      TAT      Waiting Time
Process No[1]   2              0        -2
Process No[2]   2              2        0
Process No[3]   2              4        2
Average Turn Around Time: 0.000000
Average Waiting Time: 2.000000

```

## 8. Construct a C program to simulate Round Robin scheduling algorithm with C.

### PROGRAM:

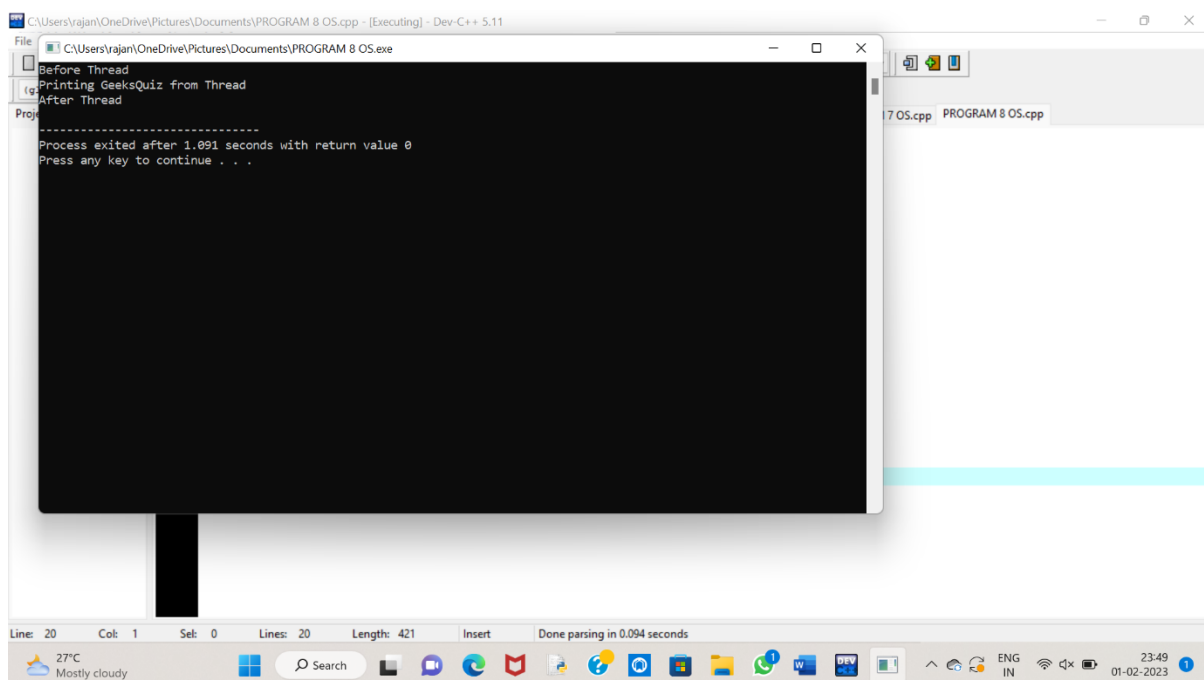
```

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
void *myThreadFun(void *vargp)
{
    sleep(1);
    printf("Printing GeeksQuiz from Thread \n");
    return NULL;
}
int main()

```

```
{  
    pthread_t thread_id;  
    printf("Before Thread\n");  
    pthread_create(&thread_id, NULL, myThreadFun, NULL);  
    pthread_join(thread_id, NULL);  
    printf("After Thread\n");  
    exit(0);  
}
```

### ***OUTPUT:***



```
File Edit View Compiler Run Window Help  
C:\Users\rajan\OneDrive\Pictures\Documents\PROGRAM 8 OS.cpp - [Executing] - Dev-C++ 5.11  
Before Thread  
Printing GeeksQuiz from Thread  
After Thread  
Process exited after 1.091 seconds with return value 0  
Press any key to continue . . .  
C:\Users\rajan\OneDrive\Pictures\Documents\PROGRAM 8 OS.exe  
PROGRAM 8 OS.cpp  
PROGRAM 8 OS.cpp  
Line: 20 Col: 1 Sel: 0 Lines: 20 Length: 421 Insert Done parsing in 0.094 seconds  
27°C Mostly cloudy 23:49 01-02-2023
```