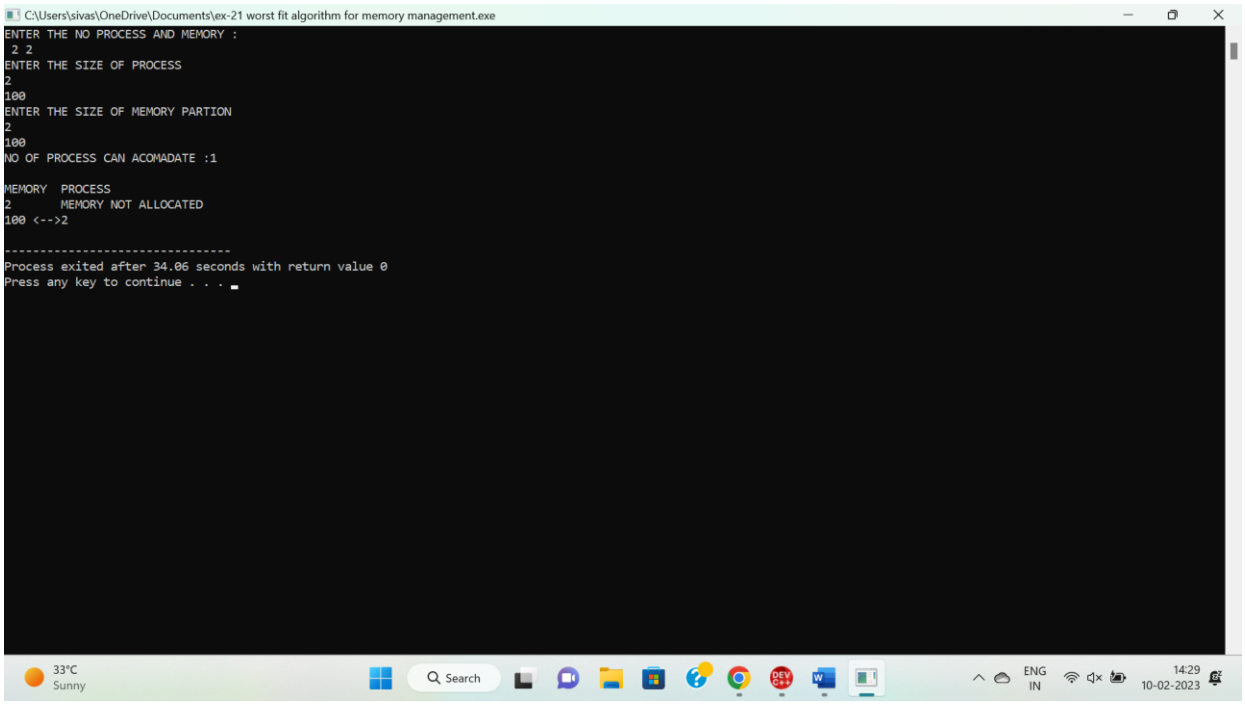
21. Develop a C program to implement worst fit algorithm of memory management.

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
struct p{
int acum[100];
int jp[100];
}st;
int main()
{
int n,m,i,count=0,j,pn[100];
int p[100],size[100];
bool flag[100];
printf("ENTER THE NO PROCESS AND MEMORY :\n ");
scanf("%d%d",&n,&m);
printf("ENTER THE SIZE OF PROCESS \n");
for(i=0;i<n;i++)
scanf("%d",&p[i]);
printf("ENTER THE SIZE OF MEMORY PARTION \n");
for(i=0;i<m;i++)
{
scanf("%d",&size[i]);
flag[i]=0;
int k;
st.acum[i]=size[i];
st.jp[i]=i;
for(k=i;k>0;k--)
{
if(st.acum[k]<=st.acum[k-1])
```

```
{
int temp=st.acum[k];
st.acum[k]=st.acum[k-1];
st.acum[k-1]=temp;
temp=st.jp[k];
st.jp[k]=st.jp[k-1];
st.jp[k-1]=temp;
}
}
}
int x=m-1;
for(i=0;i<n;i++)
{
if(p[i]<=st.acum[x]&&flag[st.jp[x]]==0)
{
flag[st.jp[x]]=true;
pn[st.jp[x]]=i;
x--;
count++;
}
}
printf("NO OF PROCESS CAN ACOMADATE :%d\n\n",count);
printf("MEMORY\tPROCESS\n");
for(i=0;i<m;i++ )
{
if(flag[i]==1)
{
printf("%d <-->%d\n",size[i],p[pn[i]]);
}
```

```
else
printf("%d\tMEMORY NOT ALLOCATED\n",size[i]);
}
return 0;
}
```



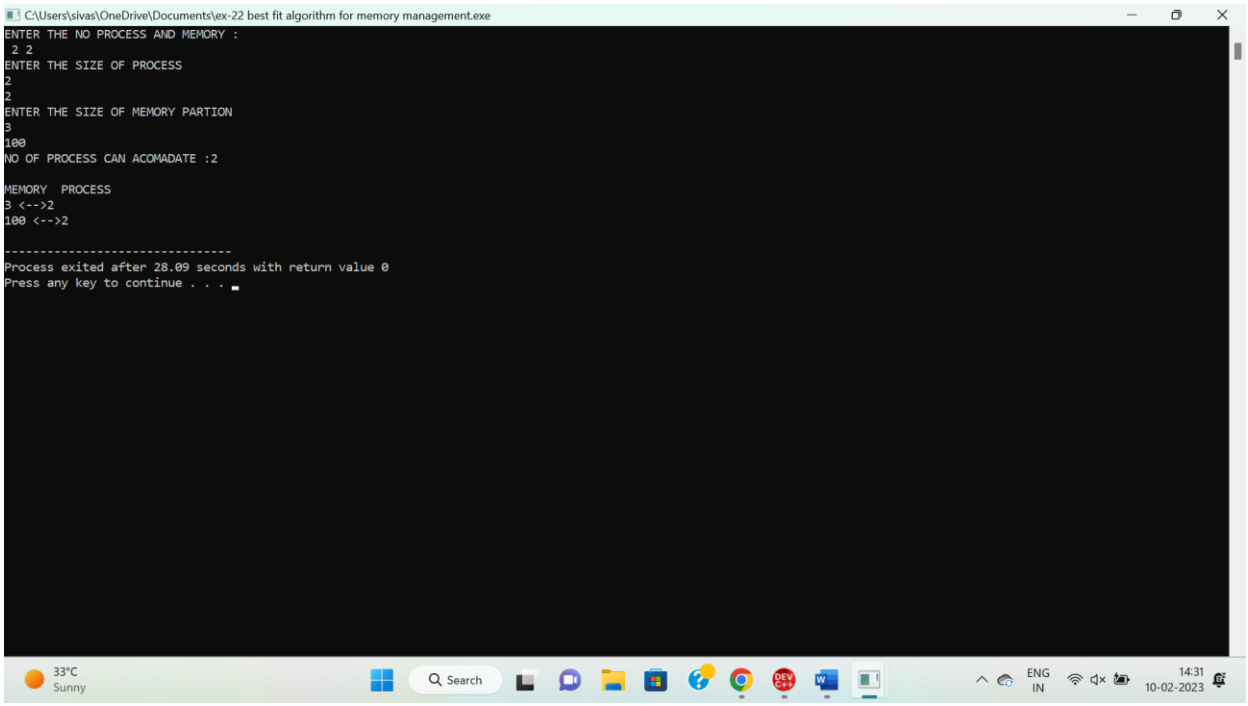
22. Construct a C program to implement best fit algorithm of memory management.

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
struct p{
int acum[100];
int jp[100];
}st;
int main()
{
int n,m,i,count=0,j,pn[100];
```

```
int p[100],size[100];
bool flag[100];
printf("ENTER THE NO PROCESS AND MEMORY :\n ");
scanf("%d%d",&n,&m);
printf("ENTER THE SIZE OF PROCESS \n");
for(i=0;i<n;i++)
scanf("%d",&p[i]);
printf("ENTER THE SIZE OF MEMORY PARTION \n");
for(i=0;i<m;i++)
{
scanf("%d",&size[i]);
flag[i]=0;
}
for(i=0;i<n;i++)
{
int ic=0,in=0;
for(j=0;j<m;j++)
{

if(p[i]<=size[j]&&flag[j]==0)
{

int k;
st.acum[in]=size[j];
st.jp[in]=j;
in++;
ic++;
for(k=ic-1;k>0;k--)
{
```

```
if(st.acum[k]<=st.acum[k-1])
{
int temp=st.acum[k];
st.acum[k]=st.acum[k-1];
st.acum[k-1]=temp;
temp=st.jp[k];
st.jp[k]=st.jp[k-1];
st.jp[k-1]=temp;
}
}
}
}
if(ic>0)
{
j=st.jp[0];
flag[j]=true;
pn[j]=i;
count++;
}
}
printf("NO OF PROCESS CAN ACOMADATE :%d\n\n",count);
printf("MEMORY\tPROCESS\n");
for(i=0;i<m;i++ )
{
if(flag[i]==1)
{
printf("%d <-->%d\n",size[i],p[pn[i]]);
}
else
```

```
printf("%d\tMEMORY NOT ALLOCATED\n",size[i]);
}
return 0;
}
```

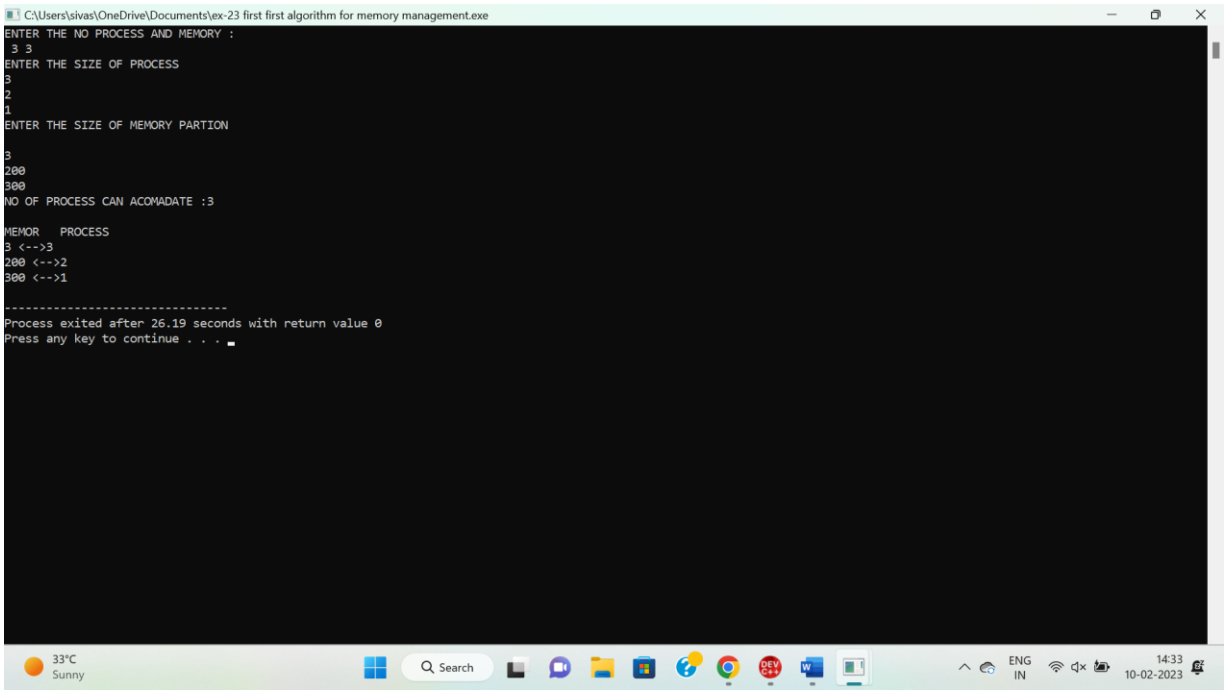


23. Construct a C program to implement first fit algorithm of memory management.

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
int main()
{
int n,m,i,count=0,j,pn[100];
int p[100],size[100];
bool flag[100];
printf("ENTER THE NO PROCESS AND MEMORY :\n ");
scanf("%d%d",&n,&m);
printf("ENTER THE SIZE OF PROCESS \n");
for(i=0;i<n;i++)
```

```
scanf("%d",&p[i]);
printf("ENTER THE SIZE OF MEMORY PARTION \n\n");
for(i=0;i<m;i++)
{
scanf("%d",&size[i]);
flag[i]=0;
}
for(i=0;i<n;i++)
{
for(j=0;j<m;j++)
{
if(p[i]<=size[j]&&flag[j]==0)
{
flag[j]=true;
pn[j]=i;
count++;
j=j+m;
}
}
}
printf("NO OF PROCESS CAN ACOMADATE :%d\n\n",count);
printf("MEMOR\tPROCESS\n");
for(i=0;i<m;i++ )
{
if(flag[i]==1)
{
printf("%d <-->%d\n",size[i],p[pn[i]]);
}
else
```

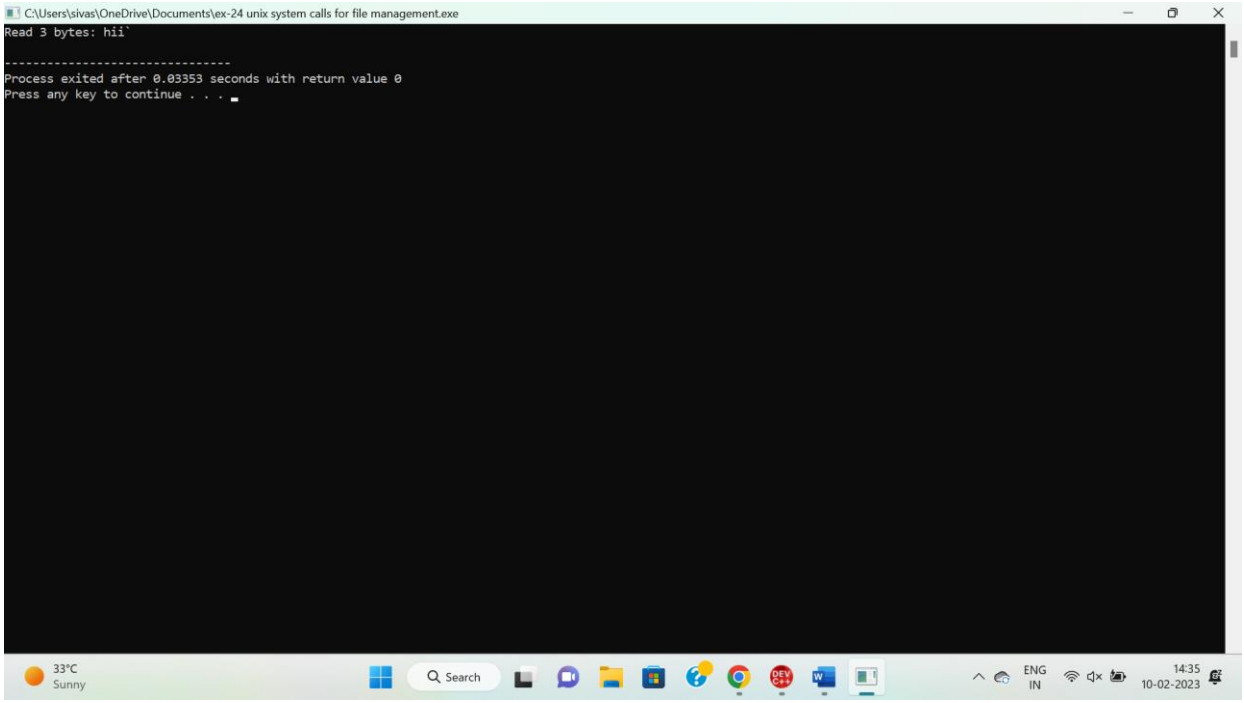
```
printf("%d\tMEMORY NOT ALLOCATED\n",size[i]);
}
return 0;
}
```



24. Design a C program to demonstrate UNIX system calls for file management.

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
int main()
{
        int fd, nbytes;
        char buffer[1024];
        fd = open("sample.txt", O_RDONLY);
        if (fd == -1) {
                perror("open");
                return 1;
```

```
        }
        nbytes = read(fd, buffer, sizeof(buffer));
        if (nbytes == -1) {
                perror("read");
                return 1;
        }
        printf("Read %d bytes: %s\n", nbytes, buffer);
        if (close(fd) == -1) {
                perror("close");
                return 1;
        }
        return 0;
}
```

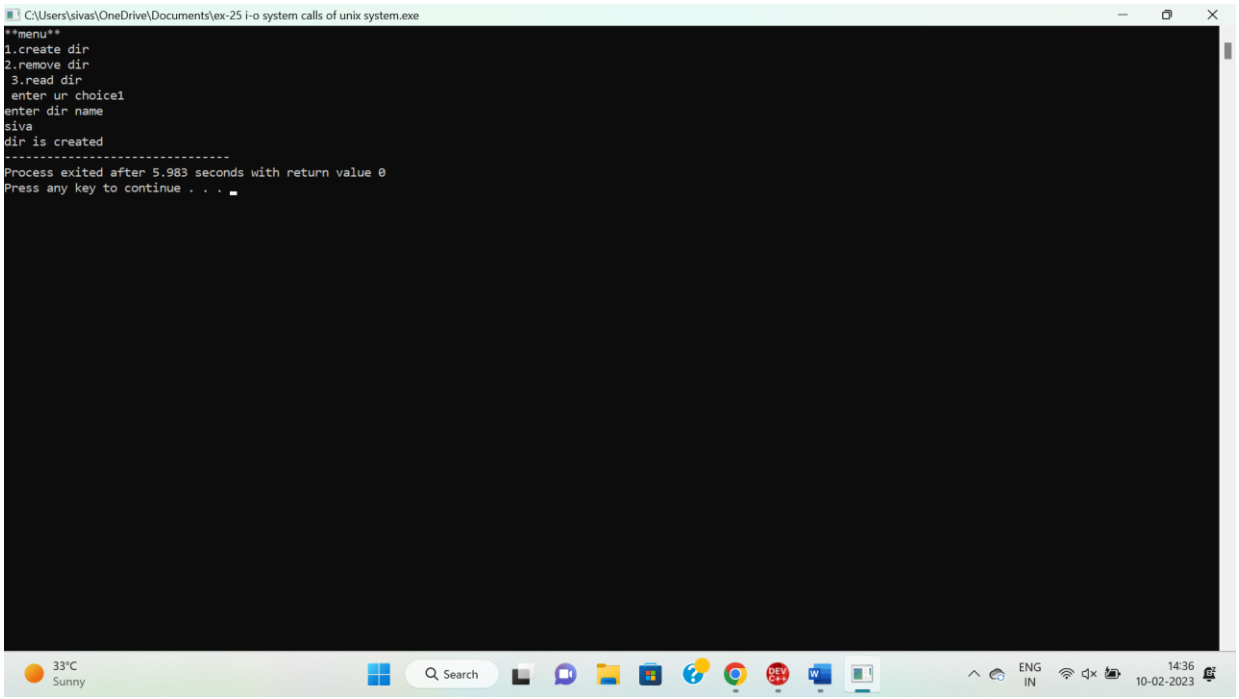


25. Construct a C program to implement the I/O system calls of UNIX (fcntl, seek, stat, opendir, readdir)

```
#include<stdio.h>
#include<fcntl.h>
```

```
#include<dirent.h>
int main()
{
char d[10]; int c,op; DIR *e;
struct dirent *sd;
printf("**menu**\n1.create dir\n2.remove dir\n 3.read dir\n enter ur choice");
scanf("%d",&op);
switch(op)
{
case 1:
{
        printf("enter dir name\n");
        scanf("%s",&d);
        c=mkdir(d);
        if(c==1)
        printf("dir is not created");
        else
        printf("dir is created");
        break;
}
case 2:
{
        printf("enter dir name\n"); scanf("%s",&d);
        c=rmdir(d);
        if(c==1)
        printf("dir is not removed");
        else
        printf("dir is removed");
        break;
```

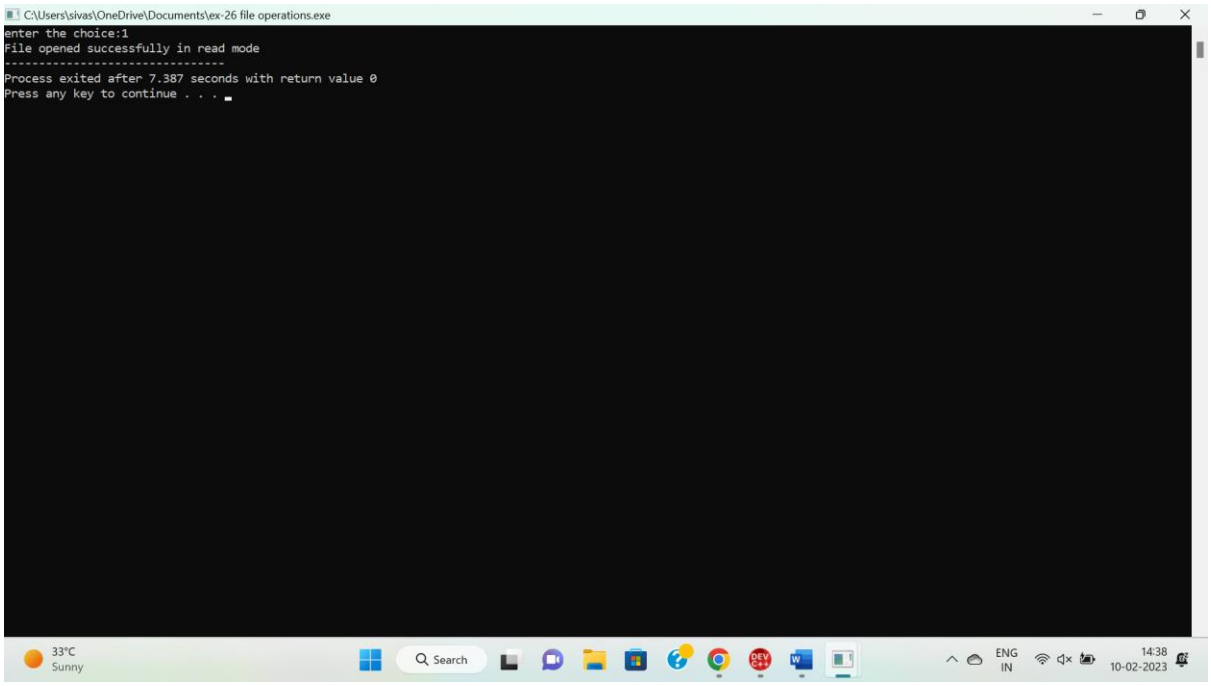
```
}
case 3:
{
        printf("enter dir name to open");
        scanf("%s",&d);
        e=opendir(d);
        if(e==NULL)
        printf("dir does not exist");
        else
        {
        printf("dir exist\n"); while((sd=readdir(e))!=NULL) printf("%s\t",sd->d_name);
        }
        closedir(e);
        break;
}
}
return 0;
}
```



26. Construct a C program to implement the file management operations.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
	int n;
	printf("enter the choice:");
	scanf("%d",&n);
	switch(n)
	{
	case 1:
		{
			FILE * file;
			if (file = fopen("sample.txt", "r"))
			{
		printf("File opened successfully in read mode");
			}
			else
			printf("The file is not present! cannot create a new file using r mode");
			fclose(file);
		}
		break;
	case 2:
		{
			FILE * file;
			if (file = fopen("sample.txt", "w"))
			{
		printf("File opened successfully in write mode or a new file is created");
```

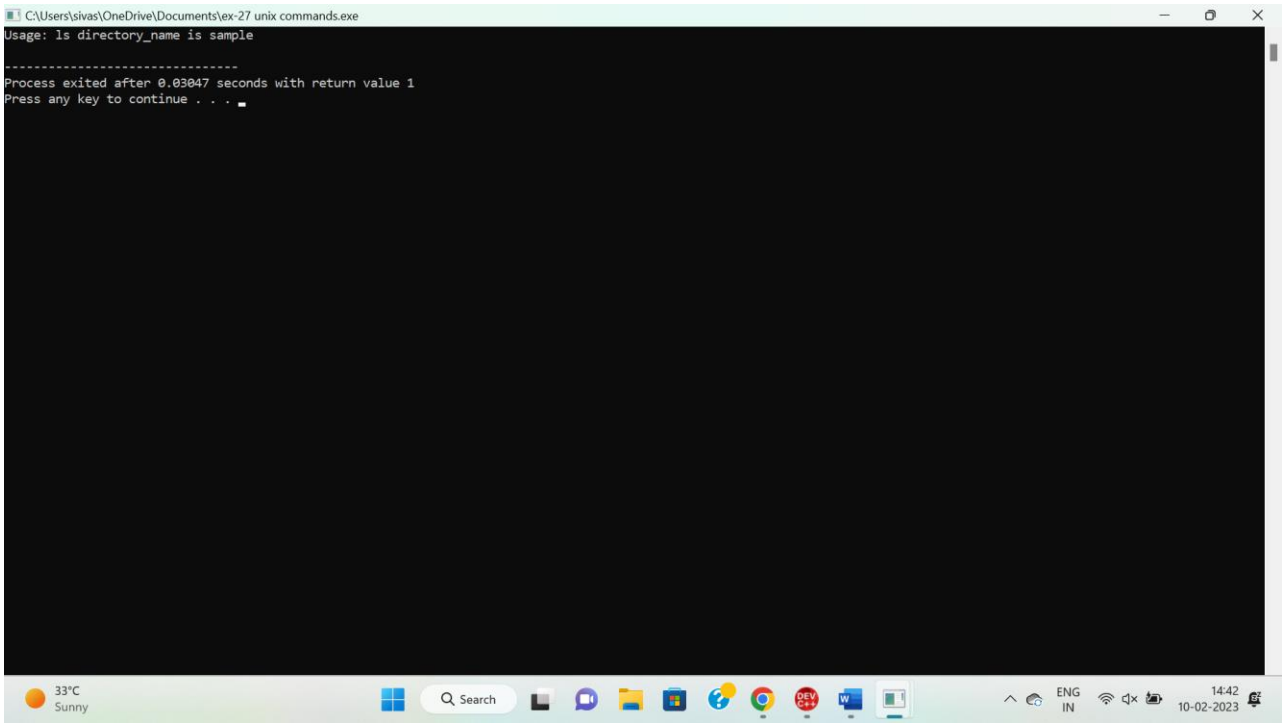
```
            }
                    else
                    printf("Error!");
                    fclose(file);
            }
            break;
        case 3:
            {
                    FILE * file;
                    char str[500];
                    if (file = fopen("sample.txt", "r"))
                      {
        while(fscanf(file,"%s", str)!=EOF)
                    {
        printf("%s", str);
            }
                    }
                    else
                    printf("Error!");
                    fclose(file);
            }
            break;
        }
            return 0;
}
```



27. Developa C program for simulatingthe function of ls UNIX Command.

```
#include <dirent.h>
#include <stdio.h>
int main(int argc, char *argv[]) {
   DIR *dp;
   struct dirent *dirp;
   if (argc != 2) {
      printf("Usage: ls directory_name is sample\n");
      return 1;
   }
   if ((dp = opendir(argv[1])) == NULL) {
      printf("Cannot open directory %s\n", argv[1]);
      return 1;
   }
   while ((dirp = readdir(dp)) != NULL) {
      printf("%s\n", dirp->d_name);
```

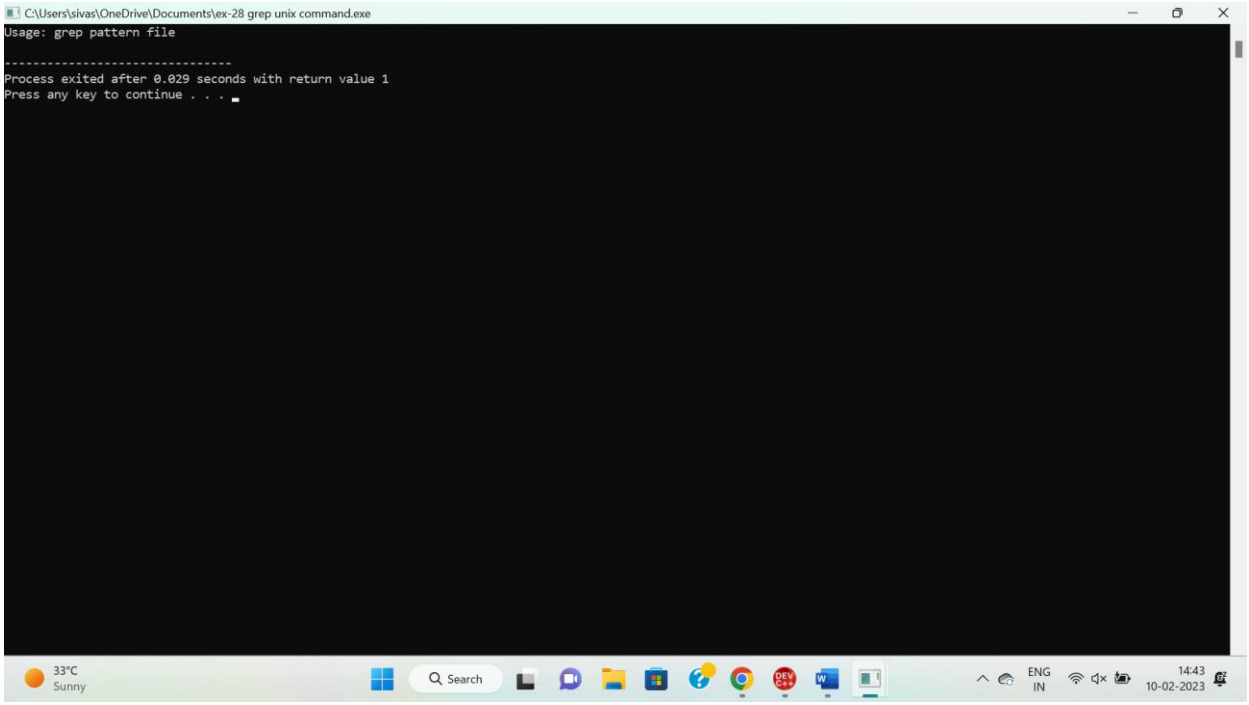
```
    }
    closedir(dp);
    return 0;
}
```



28. Write a C program for simulation of GREPUNIX command

```
#include <stdio.h>
#include <string.h>
#define MAX_LINE_LEN 1024
int main(int argc, char *argv[]) {
    char line[MAX_LINE_LEN];
    FILE *fp;
    if (argc != 3) {
        printf("Usage: grep pattern file\n");
        return 1;
    }
```

```
    fp = fopen(argv[2], "r");
    if (fp == NULL) {
        printf("open file %s\n", argv[2]);
        return 1;
    }
    while (fgets(line, MAX_LINE_LEN, fp) != NULL) {
        if (strstr(line, argv[1]) != NULL) {
            printf("%s", line);
        }
    }
    fclose(fp);
    return 0;
}
```



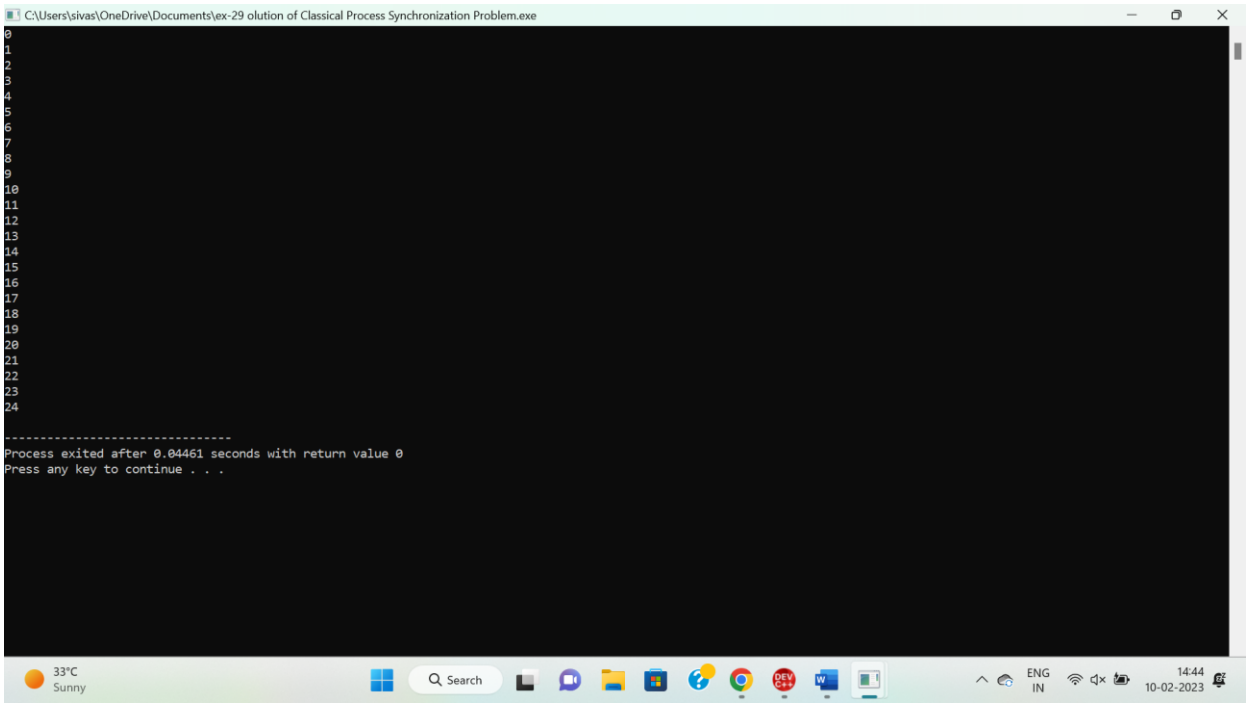
29.Write a C program to simulate the solution of Classical Process Synchronization Problem

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
```

```
#include <unistd.h>
#define BUFFER_SIZE 10
int buffer[BUFFER_SIZE];
int fill = 0;
int use = 0;
int count = 0;
sem_t empty;
sem_t full;
pthread_mutex_t mutex;
void put(int value) {
   buffer[fill] = value;
   fill = (fill + 1) % BUFFER_SIZE;
   count++;
}
int get() {
   int tmp = buffer[use];
   use = (use + 1) % BUFFER_SIZE;
   count--;
   return tmp;
}
void *producer(void *arg) {
   int i;
   for (i = 0; i < 25; i++) {
      sem_wait(&empty);
      pthread_mutex_lock(&mutex);
      put(i);
      pthread_mutex_unlock(&mutex);
      sem_post(&full);
   }
```

```
    return NULL;
}
void *consumer(void *arg) {
    int i;
    for (i = 0; i < 25; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int tmp = get();
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
        printf("%d\n", tmp);
    }
    return NULL;
}
int main() {
    pthread_t producer_thread;
    pthread_t consumer_thread;

    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    pthread_mutex_init(&mutex, NULL);

    pthread_create(&producer_thread, NULL, producer, NULL);
    pthread_create(&consumer_thread, NULL, consumer, NULL);

    pthread_join(producer_thread, NULL);
    pthread_join(consumer_thread, NULL);

    sem_destroy(&empty);
```

```
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```



30. Write C programs to demonstrate the following thread related concepts.

(i) create (ii) join (iii) equal (iv) exit

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
void* func(void* arg)
{
        pthread_detach(pthread_self());
    printf("Inside the thread\n");
        pthread_exit(NULL);
}
void fun()
{
```

```
        pthread_t ptid;
        pthread_create(&ptid, NULL, &func, NULL);
        printf("This line may be printed"
                " before thread terminates\n");
        if(pthread_equal(ptid, pthread_self()))
                printf("Threads are equal\n");
        else
                printf("Threads are not equal\n");
        pthread_join(ptid, NULL);

        printf("This line will be printed"
                " after thread ends\n");
        pthread_exit(NULL);
}
int main()
{
        fun();
        return 0;
}
```