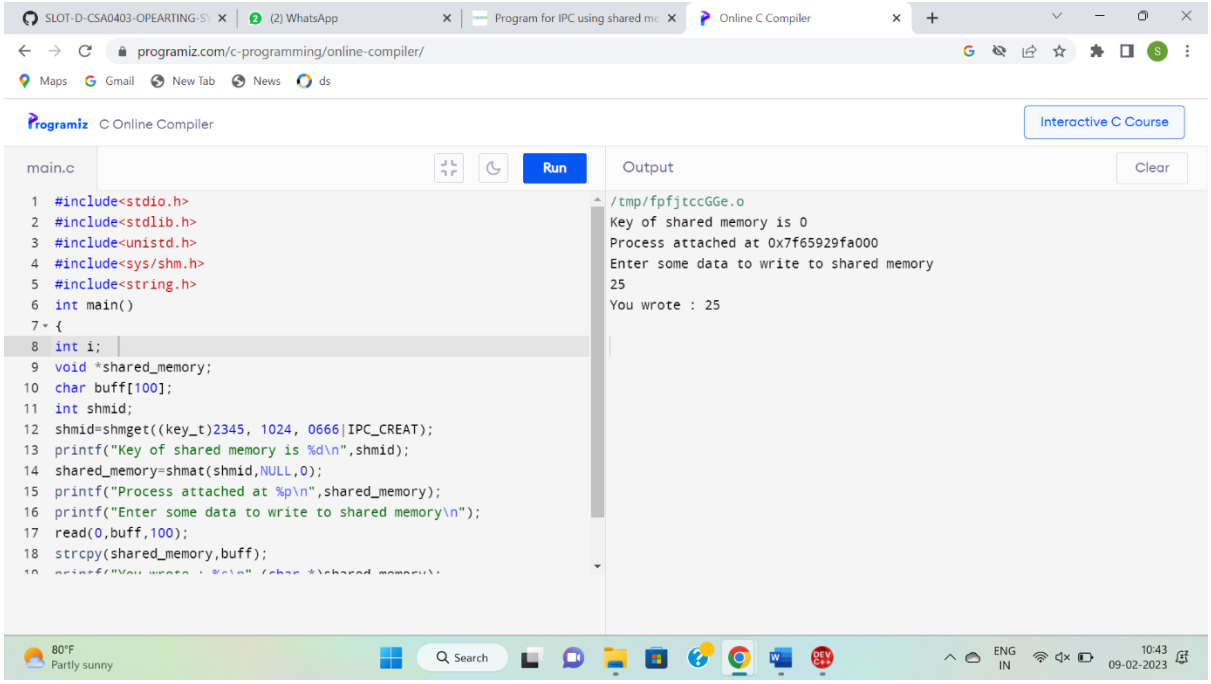
9.Illustrate the concept of inter-process communication using shared memory with a C program.

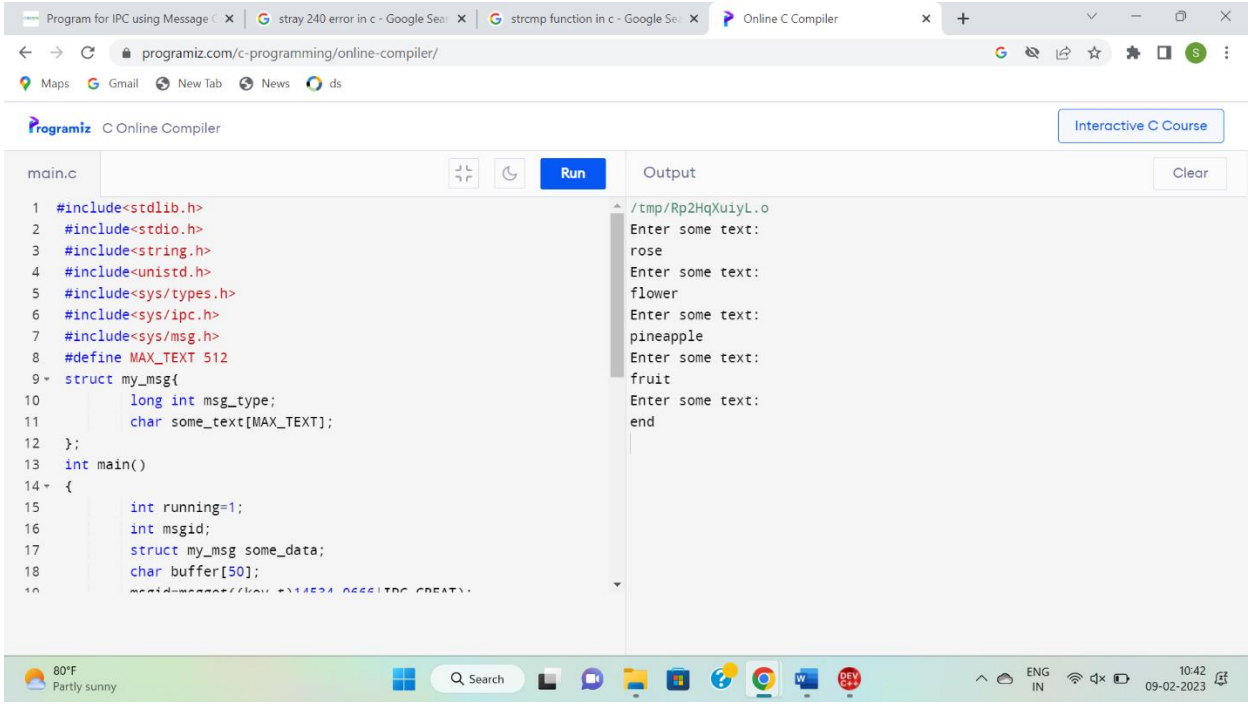
```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
int i;
void *shared_memory;
char buff[100];
int shmid;
shmid=shmget((key_t)2345, 1024, 0666|IPC_CREAT);
printf("Key of shared memory is %d\n",shmid);
shared_memory=shmat(shmid,NULL,0);
printf("Process 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);
}
```



10. Illustrate the concept of inter-process communication using message queue with a C program.

```
#include<stdlib.h>
 #include<stdio.h>
 #include<string.h>
 #include<unistd.h>
 #include<sys/types.h>
 #include<sys/ipc.h>
 #include<sys/msg.h>
 #define MAX_TEXT 512
 struct my_msg{
        long int msg_type;
        char some_text[MAX_TEXT];
 };
 int main()
 {
```

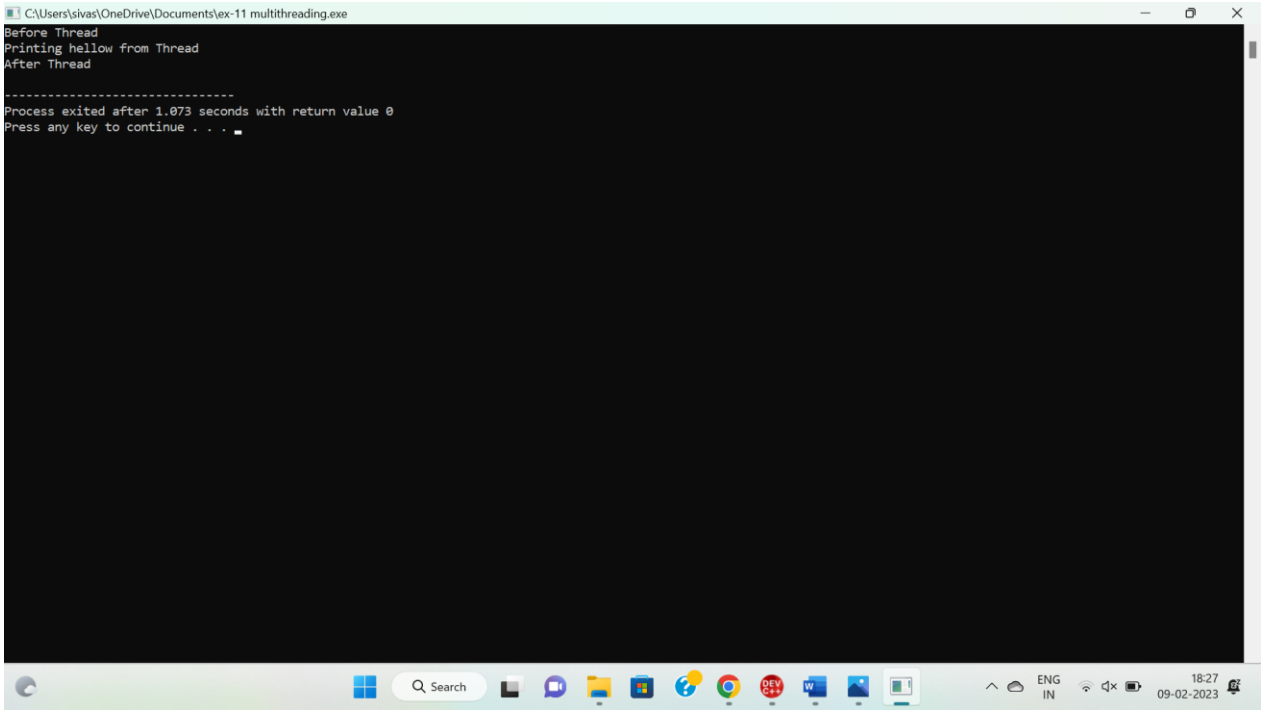
```
    int running=1;
    int msgid;
    struct my_msg some_data;
    char buffer[50];
    msgid=msgget((key_t)14534,0666|IPC_CREAT);
    if (msgid == -1)
    {
        printf("Error in creating queue\n");
        exit(0);
    }
    while(running)
    {
        printf("Enter some text:\n");
        fgets(buffer,50,stdin);
        some_data.msg_type=1;
        strcpy(some_data.some_text,buffer);
        if(msgsnd(msgid,(void *)&some_data, MAX_TEXT,0)==-1)
        {
            printf("Msg not sent\n");
        }
        if(strncmp(buffer,"end",3)==0)
        {
            running=0;
        }
    }
}
```



11. Illustrate the concept of multithreading using a C program

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
void *myThreadFun(void *vargp)
{
   sleep(1);
   printf("Printing hellow 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);
}
```

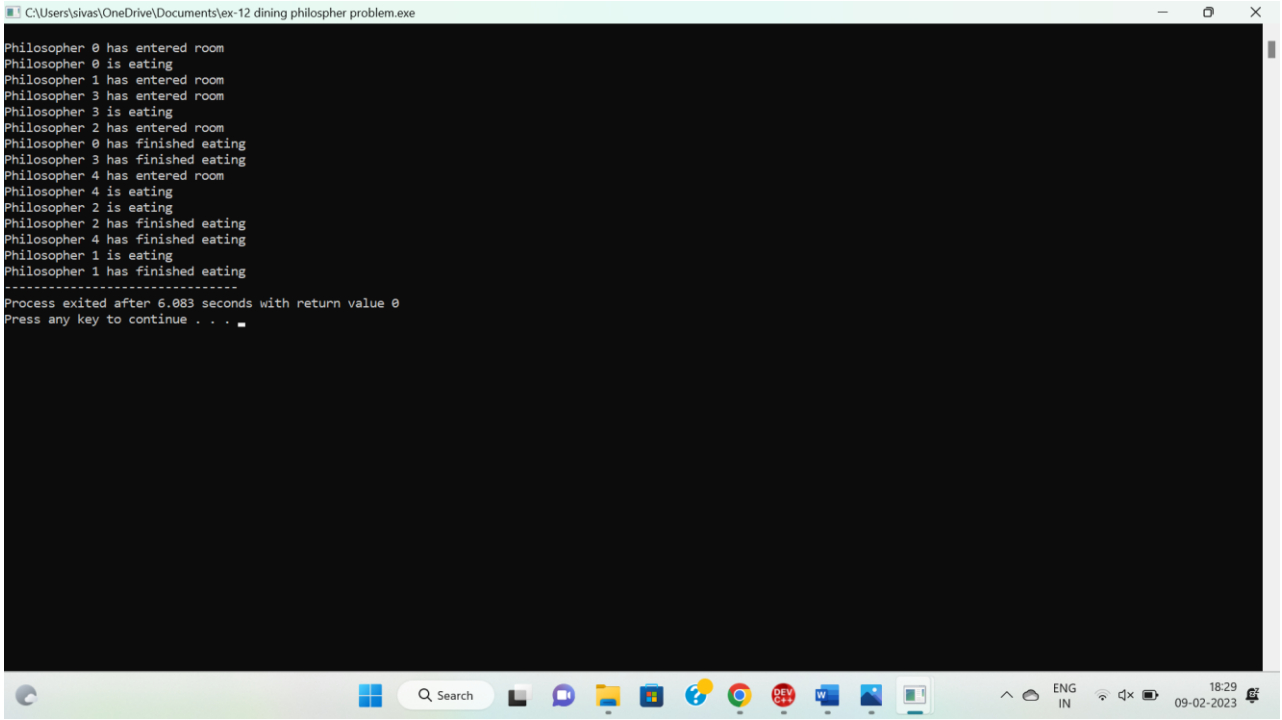


12. Design a C program to simulate the concept of Dining-Philosophers problem

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<semaphore.h>
#include<unistd.h>
sem_t room;
sem_t chopstick[5];
void * philosopher(void *);
void eat(int);
int main()
{
```

```
	int i,a[5];
	pthread_t tid[5];
	sem_init(&room,0,4);
	for(i=0;i<5;i++)
		sem_init(&chopstick[i],0,1);
	for(i=0;i<5;i++){
		a[i]=i;
		pthread_create(&tid[i],NULL,philosopher,(void *)&a[i]);
	}
	for(i=0;i<5;i++)
		pthread_join(tid[i],NULL);
		return 0;
}
void * philosopher(void * num)
{
	int phil=*(int *)num;
	sem_wait(&room);
	printf("\nPhilosopher %d has entered room",phil);
	sem_wait(&chopstick[phil]);
	sem_wait(&chopstick[(phil+1)%5]);
	eat(phil);
	sleep(2);
	printf("\nPhilosopher %d has finished eating",phil);
	sem_post(&chopstick[(phil+1)%5]);
	sem_post(&chopstick[phil]);
	sem_post(&room);
}
void eat(int phil)
{
```

```
        printf("\nPhilosopher %d is eating",phil);
}
```

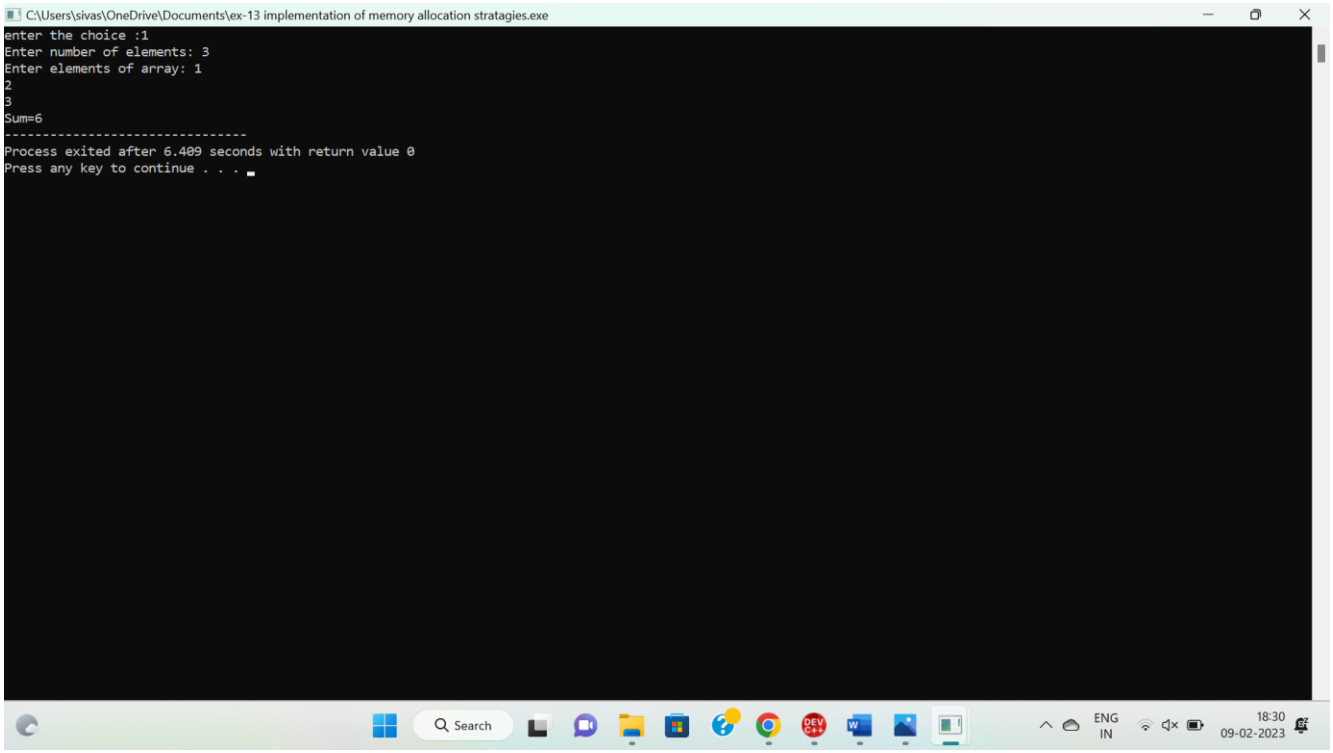


13. Construct a C program for implementation the various memory allocation strategies.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int choice;
        printf("enter the choice :");
        scanf("%d",&choice);
        switch(choice)
        {
                case 1:
                        {
                                int n,i,*ptr,sum=0;
          printf("Enter number of elements: ");
```

```
scanf("%d",&n);
ptr=(int*)malloc(n*sizeof(int));
if(ptr==NULL)
{
printf("Sorry! unable to allocate memory");
exit(0);
}
printf("Enter elements of array: ");
for(i=0;i<n;++i)
{
scanf("%d",ptr+i);
sum+=*(ptr+i);
}
printf("Sum=%d",sum);
free(ptr);
                }
                break;
        case 2:
                {
                        int n,i,*ptr,sum=0;
                        printf("Enter number of elements: ");
                scanf("%d",&n);
                ptr=(int*)calloc(n,sizeof(int));
                if(ptr==NULL)
                {
        printf("Sorry! unable to allocate memory");
        exit(0);
                }
                printf("Enter elements of array: ");
```

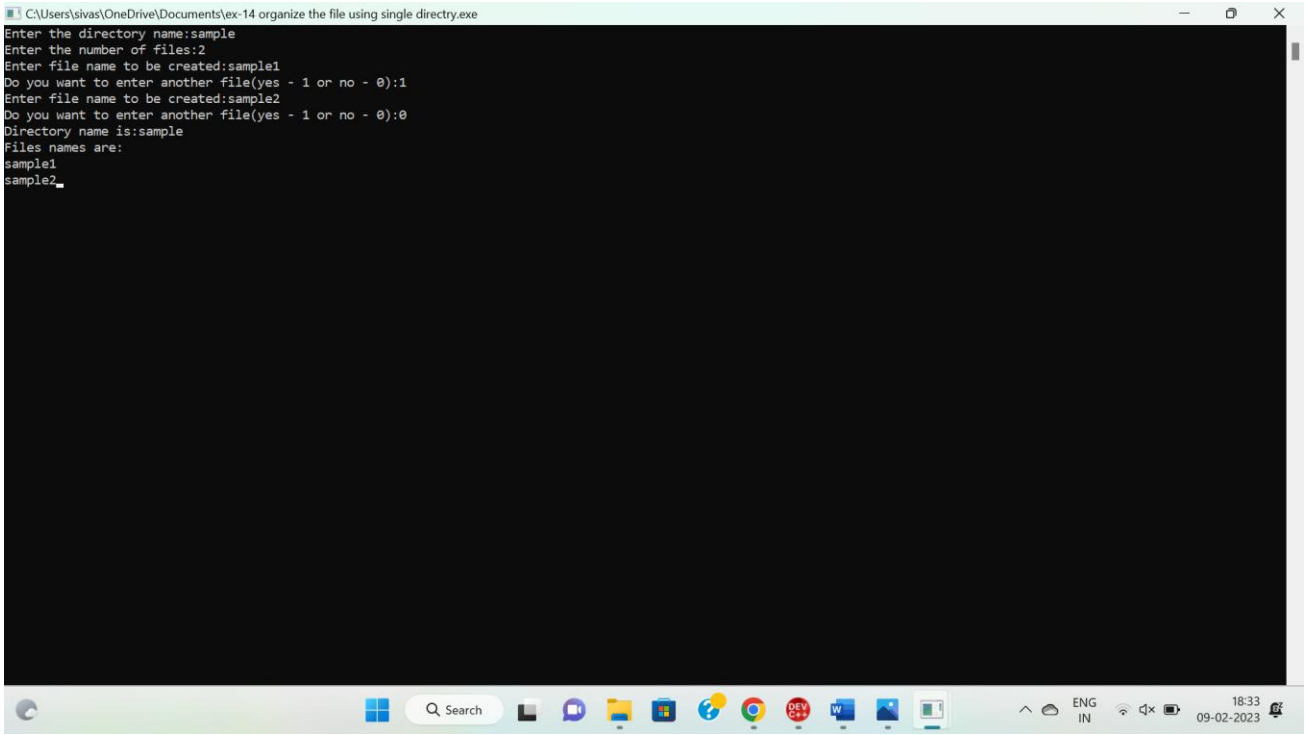
```
                for(i=0;i<n;++i)
                {
        scanf("%d",ptr+i);
        sum+=*(ptr+i);
                }
                printf("Sum=%d",sum);
                free(ptr);
                }
                break;
        default:
        {
                printf("invalid choice");
        }
    }
    return 0;
}
```



14. Construct a C program to organize the file using single level directory.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
int nf=0,i=0,j=0,ch;
char mdname[10],fname[10][10],name[10];
printf("Enter the directory name:");
scanf("%s",mdname);
printf("Enter the number of files:");
scanf("%d",&nf);
do
{
printf("Enter file name to be created:");
scanf("%s",name);
for(i=0;i<nf;i++)
{
if(!strcmp(name,fname[i]))
break;
}
if(i==nf)
{
strcpy(fname[j++],name);
nf++;
}
else
printf("There is already %s\n",name);
printf("Do you want to enter another file(yes - 1 or no - 0):");
```

```
scanf("%d",&ch);
}
while(ch==1);
printf("Directory name is:%s\n",mdname);
printf("Files names are:");
for(i=0;i<j;i++)
printf("\n%s",fname[i]);
getch();
}
```

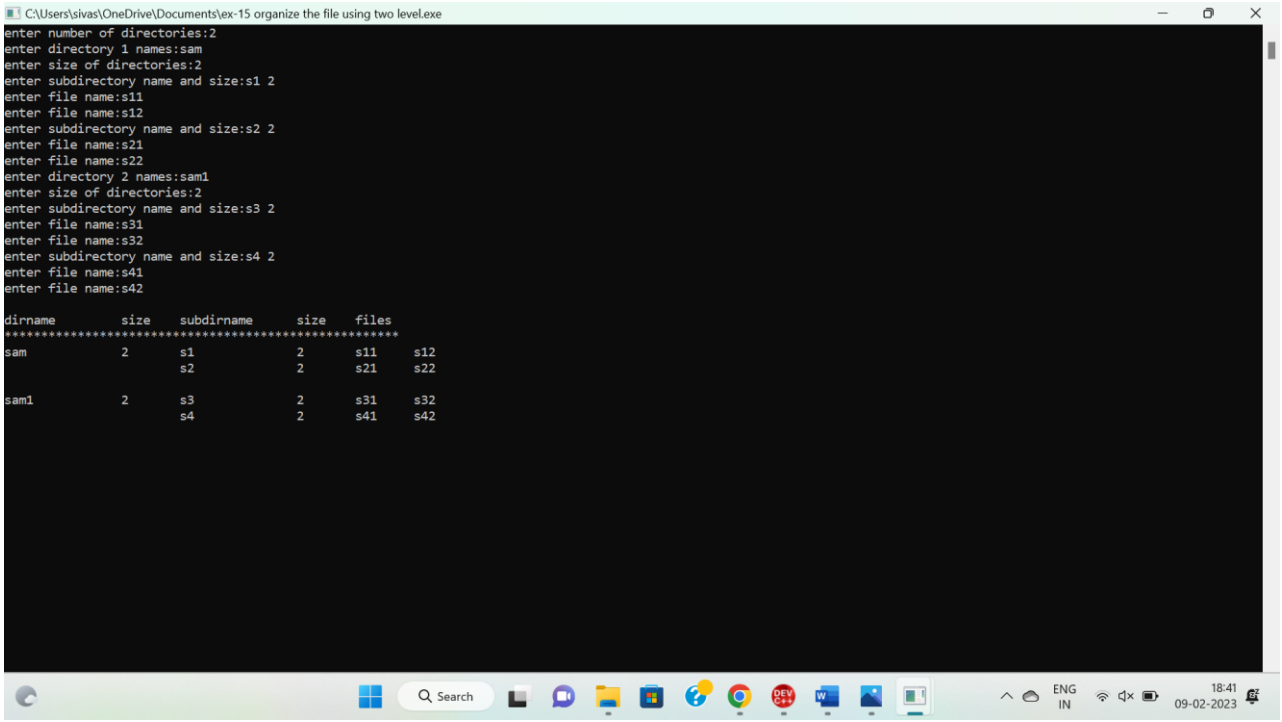


15. Design a C program to organize the file using two level directory structure.

```
#include<stdio.h>
#include<conio.h>
struct st
{
char dname[10];
```

```
char sdname[10][10];
char fname[10][10][10];
int ds,sds[10];
}dir[10];
int main()
{
int i,j,k,n;
printf("enter number of directories:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter directory %d names:",i+1);
scanf("%s",&dir[i].dname);
printf("enter size of directories:");
scanf("%d",&dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("enter subdirectory name and size:");
scanf("%s",&dir[i].sdname[j]);
scanf("%d",&dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
{
printf("enter file name:");
scanf("%s",&dir[i].fname[j][k]);
}
}
}
printf("\ndirname\t\tsize\tsubdirname\tsize\tfiles");
printf("\n*****************************************************\n");
for(i=0;i<n;i++)
{
printf("%s\t\t%d",dir[i].dname,dir[i].ds);
```

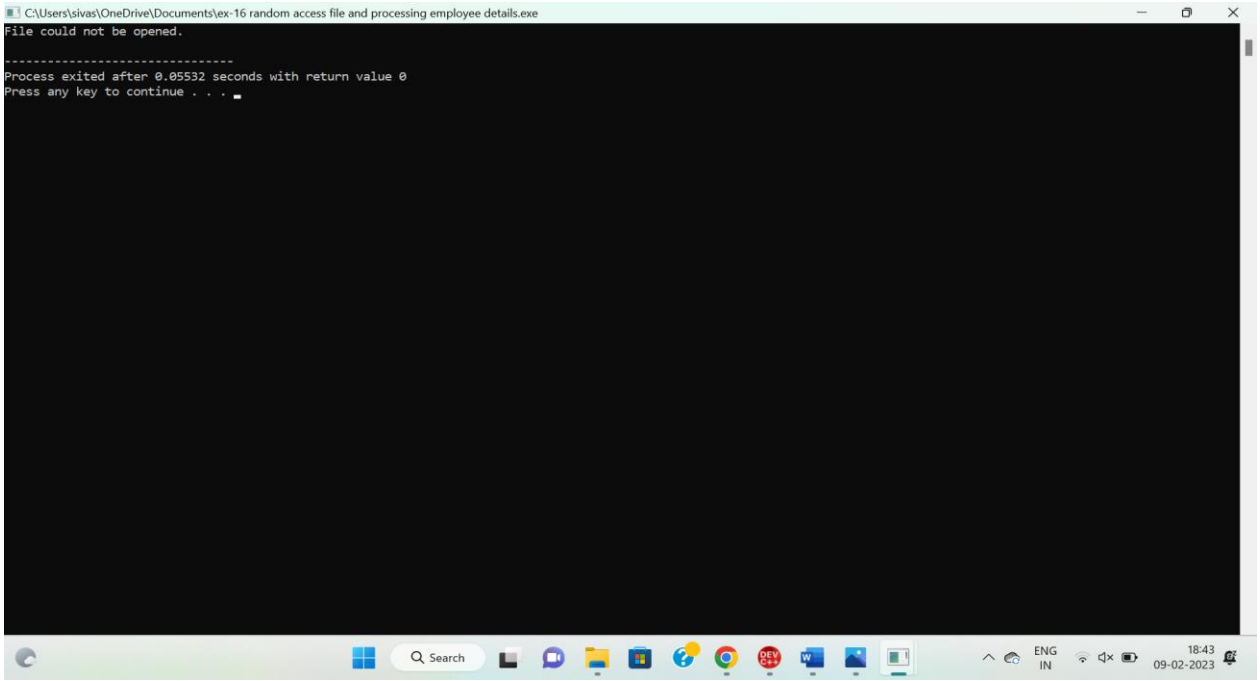
```
for(j=0;j<dir[i].ds;j++)
{
printf("\t%s\t\t%d\t",dir[i].sdname[j],dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
printf("%s\t",dir[i].fname[j][k]);
printf("\n\t\t");
}
printf("\n");
}
getch();
}
```



16. Develop a C program for implementing random access file for processing the employee details.

```
#include <stdio.h>
   struct employeedetails
            {
   unsigned int phnno;
   char lastName[ 15 ];
```

```
char firstName[ 10 ];
double salary;
};
int main(void)
        {
FILE *cfPtr;
struct employeedetails employee = { 0, "", "", 0.0 };
if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL )
        {
  puts( "File could not be opened." );
}
else
        {
  printf( "%s", "Enter phone number: ");
  scanf( "%d", &employee.phnno );
while ( employee.phnno != 0 ) {
printf( "%s", "Enter lastname, firstname,salary\n? " );
fscanf( stdin, "%14s%9s%lf", employee.lastName,employee.firstName, &employee.salary
);
  fseek( cfPtr, ( employee.phnno - 1 )*sizeof( struct employeedetails ), SEEK_SET );
  fwrite( &employee, sizeof( struct employeedetails ), 1, cfPtr );
  printf( "%s", "Enter phone number\n? " );
  scanf( "%d", &employee.phnno );
  }
  fclose( cfPtr );
  }
}
```



17. Illustrate the deadlock avoidance concept by simulating Banker's algorithm with C.

```
#include<stdio.h>
#include<conio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
int i,j;
printf("********** Banker's Algorithm ************\n");
input();
show();
```

```
cal();
getch();
return 0;
}
void input()
{
int i,j;
printf("Enter the no of Processes\t");
scanf("%d",&n);
printf("Enter the no of resources instances\t");
scanf("%d",&r);
printf("Enter the Max Matrix\n");
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
scanf("%d",&max[i][j]);
}
}
printf("Enter the Allocation Matrix\n");
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
scanf("%d",&alloc[i][j]);
}

}
printf("Enter the available Resources\n");
```

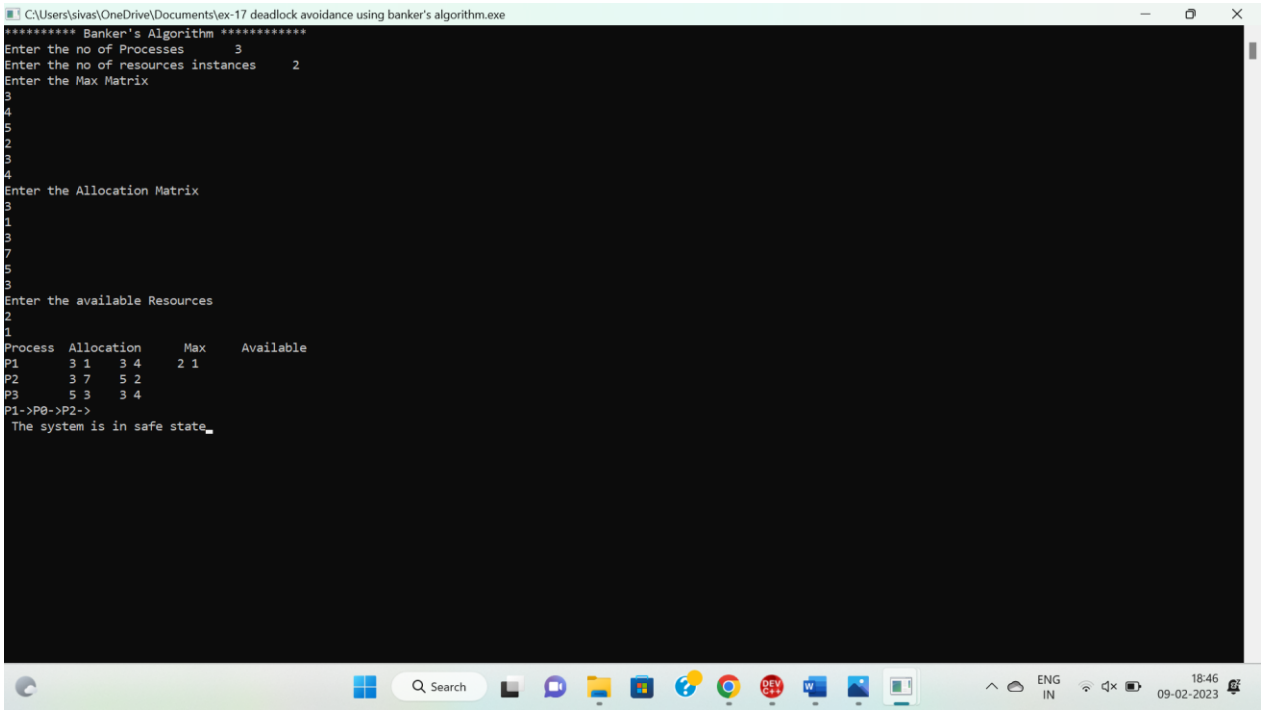
```
for(j=0;j<r;j++)
{
scanf("%d",&avail[j]);
}
}
void show()
{
int i,j;
printf("Process\t Allocation\t Max\t Available\t");
for(i=0;i<n;i++)
{
printf("\nP%d\t ",i+1);
for(j=0;j<r;j++)
{
printf("%d ",alloc[i][j]);
}
printf("\t");
for(j=0;j<r;j++)
{
printf("%d ",max[i][j]);
}
printf("\t");
if(i==0)
{
for(j=0;j<r;j++)
printf("%d ",avail[j]);
}
}
}
```

```
void cal()
{
int finish[100],temp,need[100][100],flag=1,k,c1=0;
int safe[100];
int i,j;
for(i=0;i<n;i++)
{
finish[i]=0;
}
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)

{
need[i][j]=max[i][j]-alloc[i][j];
}
}
printf("\n");
while(flag)
{
flag=0;
for(i=0;i<n;i++)
{
int c=0;
for(j=0;j<r;j++)
{
if((finish[i]==0)&&(need[i][j]<=avail[j]))
{
c++;
```

```
if(c==r)
{
for(k=0;k<r;k++)
{
avail[k]+=alloc[i][j];
finish[i]=1;
flag=1;
}
printf("P%d->",i);
if(finish[i]==1)
{
i=n;
}
}
}
}
}
}
for(i=0;i<n;i++)
{
if(finish[i]==1)
{
c1++;
}
else
{
printf("P%d->",i);
}
}
```

```
if(c1==n)
{
printf("\n The system is in safe state");
}
else
{
printf("\n Process are in dead lock");
printf("\n System is in unsafe state");
}
}
```

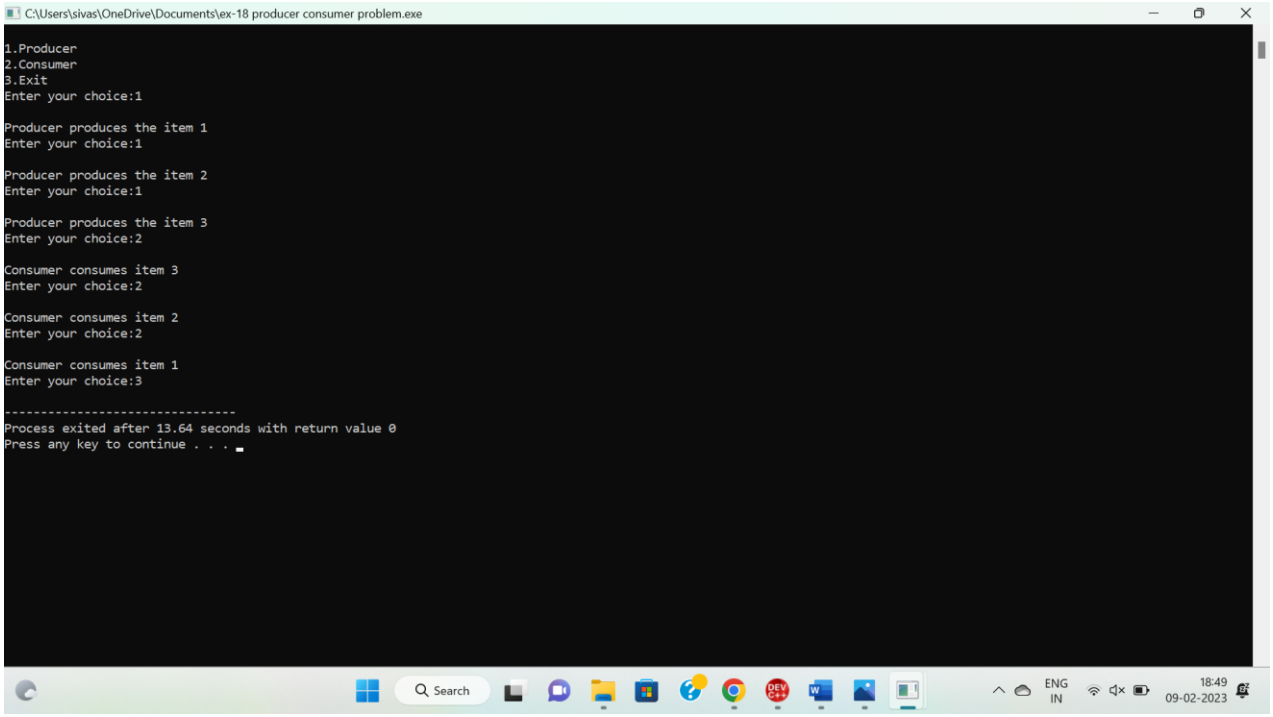


18 Construct a C program to simulate producer-consumer problem using semaphores.

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
int main()
{
   int n;
```

```
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n1.Producer\n2.Consumer\n3.Exit");
    while(1)
    {
        printf("\nEnter your choice:");
        scanf("%d",&n);
        switch(n)
        {
            case 1:    if((mutex==1)&&(empty!=0))
                           producer();
                       else
                           printf("Buffer is full!!");
                       break;
            case 2:    if((mutex==1)&&(full!=0))
                           consumer();
                       else
                           printf("Buffer is empty!!");
                       break;
            case 3:
                       exit(0);
                       break;
        }
    }
    return 0;
}
int wait(int s)
```

```
{
   return (--s);
}
int signal(int s)
{
   return(++s);
}
void producer()
{
   mutex=wait(mutex);
   full=signal(full);
   empty=wait(empty);
   x++;
   printf("\nProducer produces the item %d",x);
   mutex=signal(mutex);
}
void consumer()
{
   mutex=wait(mutex);
   full=wait(full);
   empty=signal(empty);
   printf("\nConsumer consumes item %d",x);
   x--;
   mutex=signal(mutex);
}
```

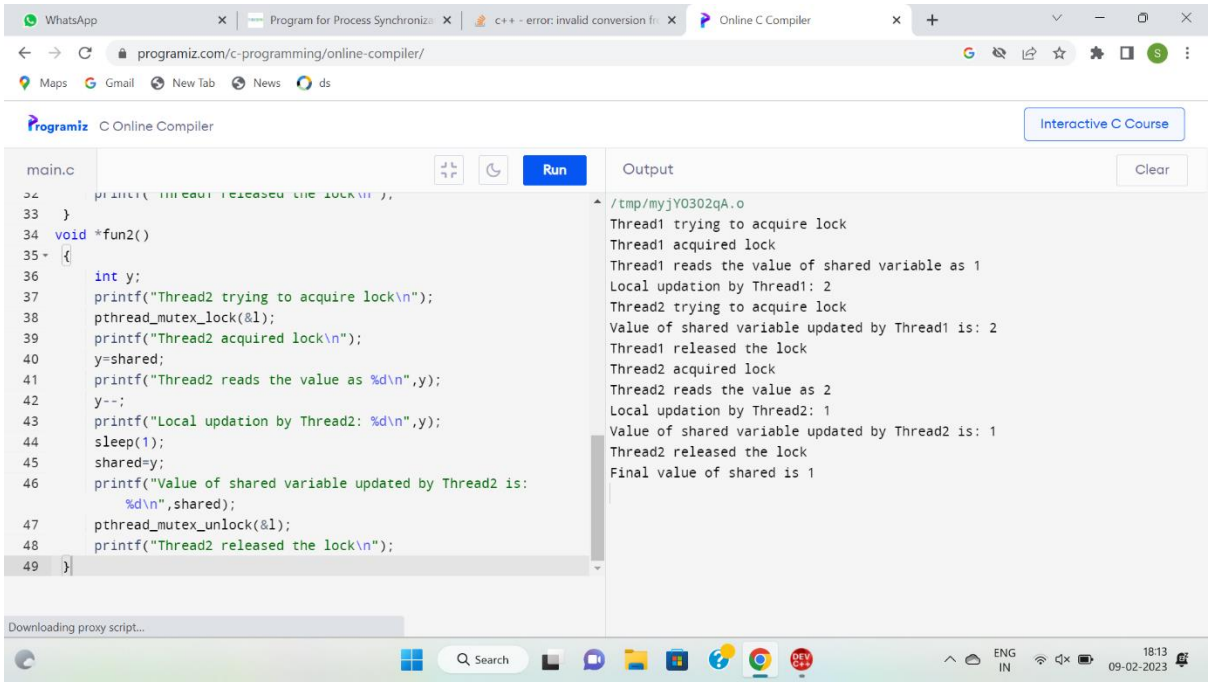


19. Design a C program to implement process synchronization using mutex locks.

```
#include<pthread.h>
#include<stdio.h>
#include<unistd.h>
void *fun1();
void *fun2();
 int shared=1;
 pthread_mutex_t l;
 int main()
 {
 pthread_mutex_init(&l, NULL);
 pthread_t thread1, thread2;
 pthread_create(&thread1, NULL, fun1, NULL);
 pthread_create(&thread2, NULL, fun2, NULL);
 pthread_join(thread1, NULL);
 pthread_join(thread2,NULL);
```

```
 printf("Final value of shared is %d\n",shared);
 }
void *fun1()
 {
    int x;
    printf("Thread1 trying to acquire lock\n");
    pthread_mutex_lock(&l);
   printf("Thread1 acquired lock\n");
    x=shared;
    printf("Thread1 reads the value of shared variable as %d\n",x);
    x++;
    printf("Local updation by Thread1: %d\n",x);
    sleep(1);
    shared=x;
    printf("Value of shared variable updated by Thread1 is: %d\n",shared);
    pthread_mutex_unlock(&l);
    printf("Thread1 released the lock\n");
 }
void *fun2()
 {
    int y;
    printf("Thread2 trying to acquire lock\n");
    pthread_mutex_lock(&l);
    printf("Thread2 acquired lock\n");
    y=shared;
    printf("Thread2 reads the value as %d\n",y);
    y--;
    printf("Local updation by Thread2: %d\n",y);
    sleep(1);
```

```
    shared=y;
    printf("Value of shared variable updated by Thread2 is: %d\n",shared);
    pthread_mutex_unlock(&l);
    printf("Thread2 released the lock\n");
}
```



20. Construct a C program to simulate Reader-Writer problem using Semaphores.

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
sem_t wrt;
pthread_mutex_t mutex;
int cnt = 1;
int numreader = 0;
void *writer(void *wno)
{
   sem_wait(&wrt);
   cnt = cnt*2;
```

```
    printf("Writer %d modified cnt to %d\n",(*((int *)wno)),cnt);
    sem_post(&wrt);
}
void *reader(void *rno)
{
    pthread_mutex_lock(&mutex);
    numreader++;
    if(numreader == 1) {
        sem_wait(&wrt);
    }
    pthread_mutex_unlock(&mutex);
    printf("Reader %d: read cnt as %d\n",*((int *)rno),cnt);
    pthread_mutex_lock(&mutex);
    numreader--;
    if(numreader == 0)
            {
        sem_post(&wrt);
    }
    pthread_mutex_unlock(&mutex);
}
int main()
{
    pthread_t read[10],write[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&wrt,0,1);
    int a[10] = {1,2,3,4,5,6,7,8,9,10};
    for(int i = 0; i < 10; i++) {
        pthread_create(&read[i], NULL, (void *)reader,(void *)&a[i]);
    }
```

```
    for(int i = 0; i < 5; i++)
            {
        pthread_create(&write[i], NULL, (void *)writer, (void *)&a[i]);
    }
    for(int i = 0; i < 10; i++)
            {?
        pthread_join(read[i], NULL);
    }
    for(int i = 0; i < 5; i++)
            {
        pthread_join(write[i], NULL);
    }
    pthread_mutex_destroy(&mutex);
    sem_destroy(&wrt);
    return 0;

}
```