NNameM.Prem swaroop

Reg no :192111627

CSA0470

OPERATING SYSTEM WITH DESIGN PRINCIPLES

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

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>

#include <unistd.h>
int main(void) { pid_t
pid = fork(); if(pid ==
0) {
printf("Child => PPID: %d PID: %d\n", getppid(), getpid()); exit(EXIT_SUCCESS);
} else if(pid > 0)
{
printf("Parent => PID: %d\n", getpid());
printf("Waiting for child process to finish.\n");
wait(NULL); printf("Child process
finished.\n");
}
else { printf("Unable to create child
process.\n");
}
```

return EXIT_SUCCESS;

}

```
/tmp/zSKkMm5Fyw.o
Parent => PID: 677
Waiting for child process to finish.
Child => PPID: 677 PID: 678
Child process finished.
```

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

#include <stdio.h>

#include <stdlib.h> int

main()

{

FILE *fptr1, *fptr2; char filename[100], c;

printf("Enter the filename to open for reading \n");

scanf("%s", filename); fptr1 = fopen(filename, "r");

if (fptr1 == NULL)

{

printf("Cannot open file %s \n", filename); exit(0);

}

printf("Enter the filename to open for writing \n");

scanf("%s", filename); fptr2 = fopen(filename,

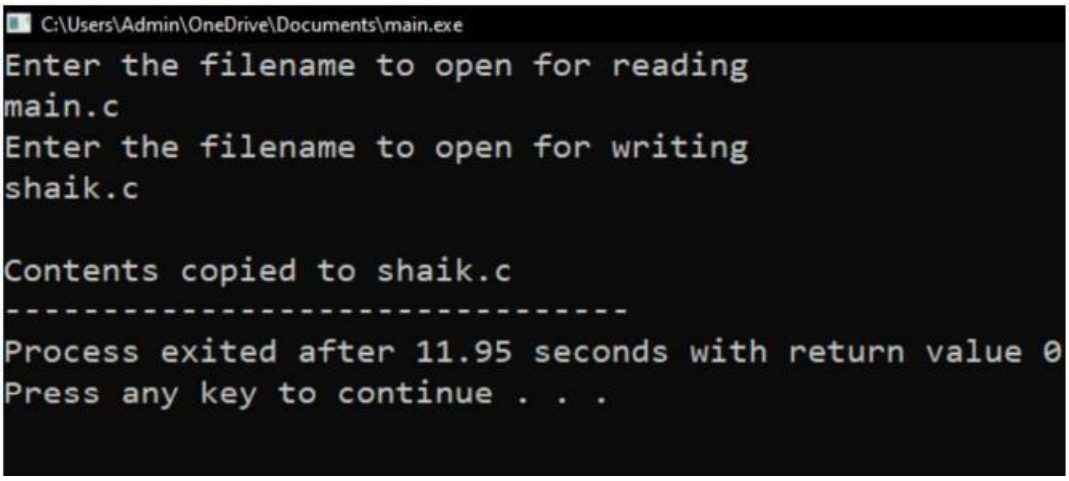
"w"); if (fptr2 == NULL)

{

printf("Cannot open file %s \n", filename); exit(0);

}

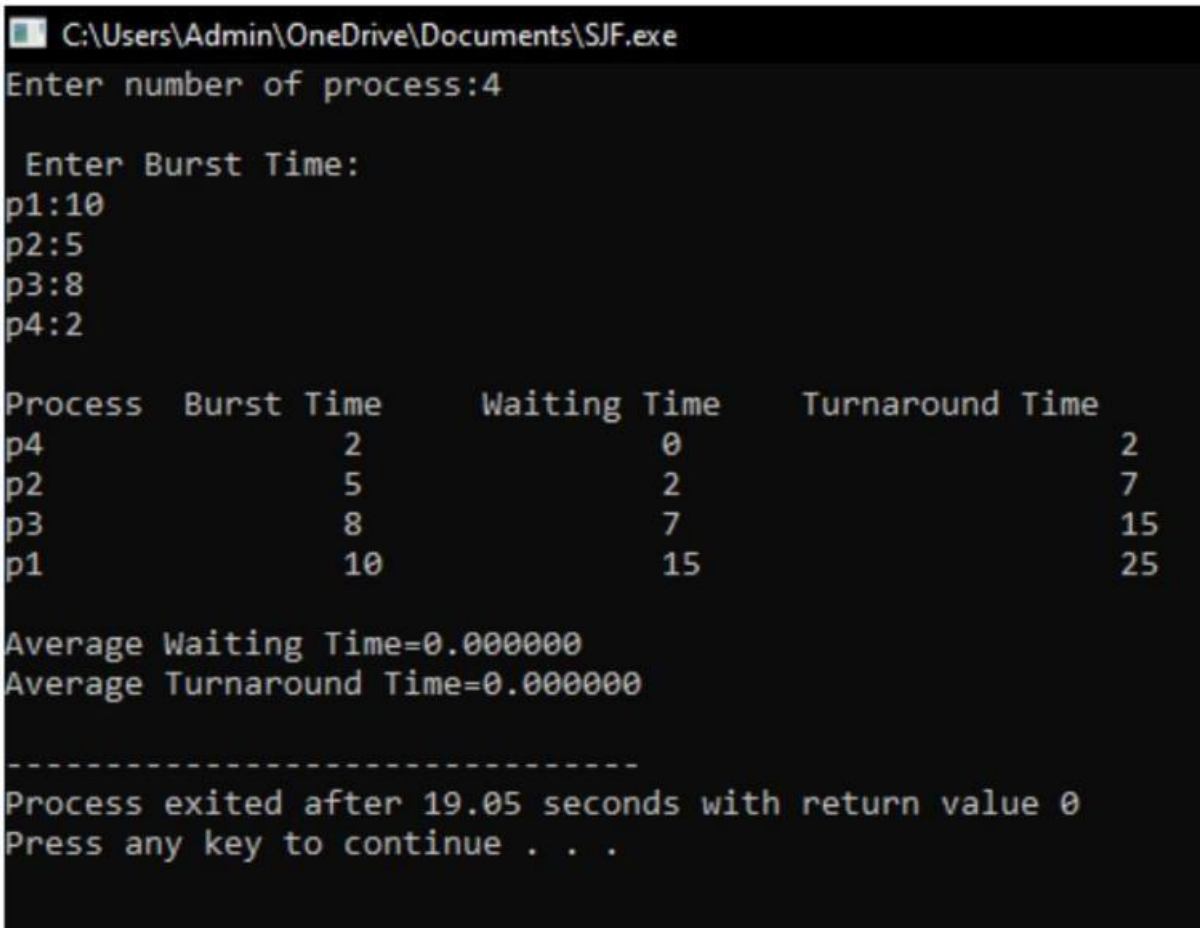
```
c = fgetc(fptr1); while
(c != EOF)
{
fputc(c, fptr2);
c = fgetc(fptr1);
}
printf("\nContents copied to %s", filename);
fclose(fptr1); fclose(fptr2); return 0;
}
```



3.Desig n 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 #include<stdio.h> int main()

```
{
int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
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:%d",avwt);
printf("\nAverage Turnaround Time:%d",avtat);  return
0;
}
```

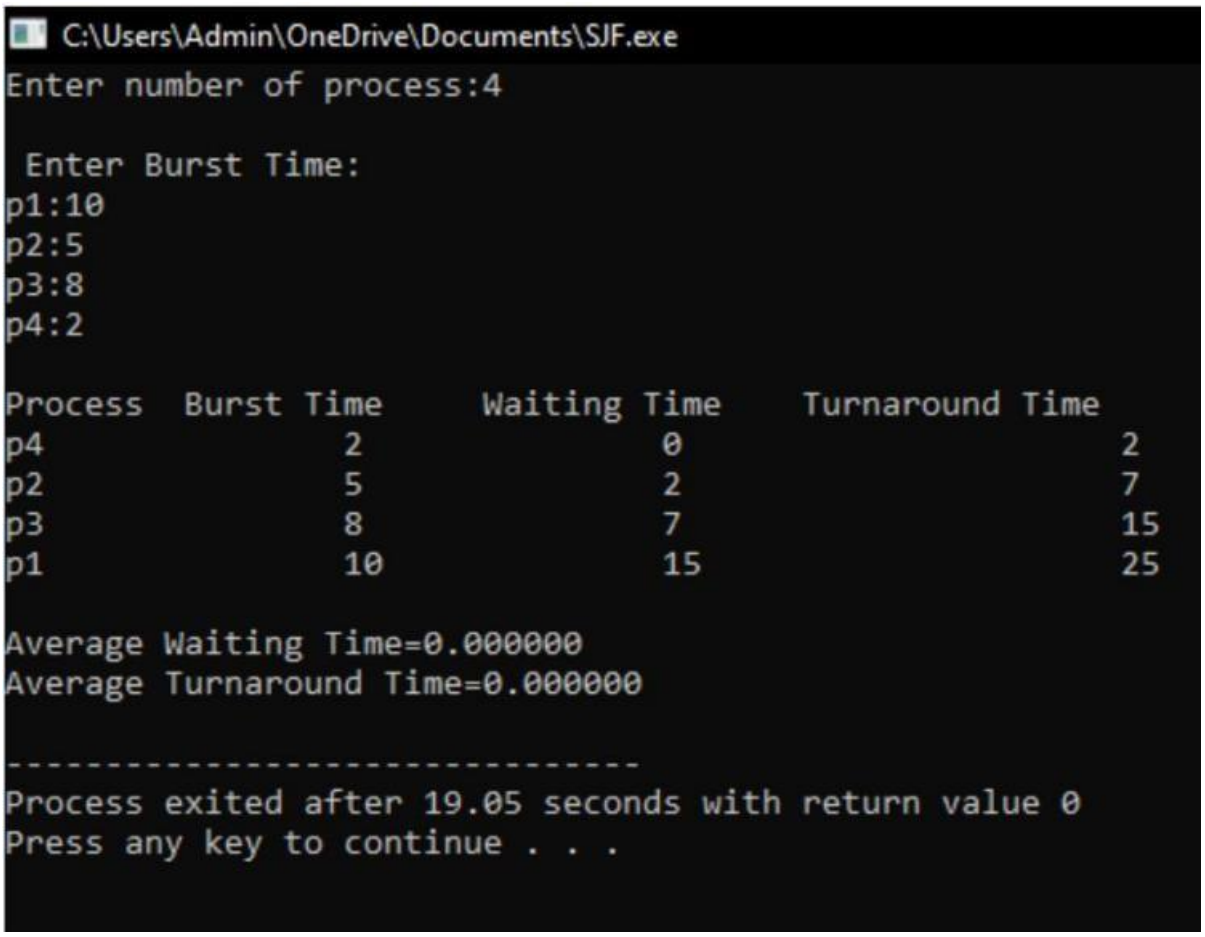


4.Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next. #include<stdio.h> int main()

```
{
 int n,r,i,j,k,p,u=0,s=0,m;  int
block[10],run[10],active[10],newreq[10];  int
max[10][10],resalloc[10][10],resreq[10][10];  int
totalloc[10],totext[10],simalloc[10];  printf("Enter the
no of processes:");  scanf("%d",&n);  printf("Enter the
no ofresource classes:");  scanf("%d",&r);  printf("Enter
the total existed resource in each class:");  for(k=1;
k<=r; k++)
 scanf("%d",&totext[k]); printf("Enter the allocated
resources:"); for(i=1; i<=n; i++) for(k=1; k<=r; k++)
scanf("%d",&resalloc);  printf("Enter the process making the
new request:");  scanf("%d",&p);  printf("Enter the
requested resource:");  for(k=1; k<=r; k++)
scanf("%d",&newreq[k]);  printf("Enter the process which
are n blocked or running:");  for(i=1; i<=n; i++)
 {
 if(i!=p)
 {
 printf("process %d:\n",i+1);
scanf("%d%d",&block[i],&run[i]);
 }
 }
 block[p]=0;
run[p]=0;  for(k=1;
k<=r; k++)
 {
 j=0;  for(i=1;
i<=n; i++)
 {
```

```
 totalloc[k]=j+resalloc[i][k];
j=totalloc[k];
 }
 }
 for(i=1; i<=n; i++)
 {
 if(block[i]==1||run[i]==1)
active[i]=1;  else
active[i]=0;
 }
 for(k=1; k<=r; k++)
 {
 resalloc[p][k]+=newreq[k];
totalloc[k]+=newreq[k];
 }
 for(k=1; k<=r; k++)
 {
 if(totext[k]-totalloc[k]<0)
 {
 u=1;
break;
 }
 }
 if(u==0)
 {
 for(k=1; k<=r; k++)
simalloc[k]=totalloc[k];
for(s=1; s<=n; s++)  for(i=1;
i<=n; i++)
 {
if(active[i]==1)
```

```
{
j=0; for(k=1; k<=r;
k++)
{ if((totext[k]-simalloc[k])<(max[i][k]-resalloc[i][k]))
{
j=1;
break;
}
 }
 }
 if(j==0)
 {
 active[i]=0;  for(k=1;
k<=r; k++)
simalloc[k]=resalloc[i][k];
 }
 }
 m=0;  for(k=1; k<=r; k++)
resreq[p][k]=newreq[k];
printf("Deadlock willn't occur");
 }
 else
 {
 for(k=1; k<=r; k++)
 {
 resalloc[p][k]=newreq[k];
totalloc[k]=newreq[k];
 }
printf("Deadlock will occur");
}
}
```



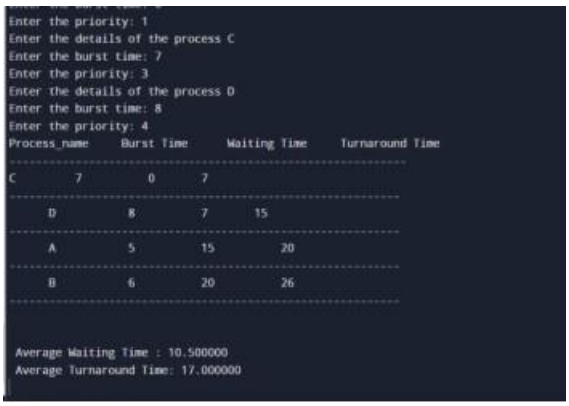
5.Illustrate the deadlock avoidance concept by simulating Banker's algorithm with C

```
#include<stdio.h> int main()
{
 int n,r,i,j,k,p,u=0,s=0,m;  int
block[10],run[10],active[10],newreq[10];  int
max[10][10],resalloc[10][10],resreq[10][10];  int
totalloc[10],totext[10],simalloc[10];  printf("Enter the
no of processes:");  scanf("%d",&n); printf("Enter the
no ofresource classes:"); scanf("%d",&r); printf("Enter
the total existed resource in each class:"); for(k=1;
k<=r; k++) scanf("%d",&totext[k]);

 printf("Enter the allocated resources:"); for(i=1; i<=n; i++)
for(k=1; k<=r; k++) scanf("%d",&resalloc); printf("Enter the
process making the new request:");  scanf("%d",&p);
printf("Enter the requested resource:");  for(k=1; k<=r; k++)
```

```
scanf("%d",&newreq[k]);  printf("Enter the process which
are n blocked or running:");
 for(i=1; i<=n; i++)
 {
 if(i!=p)
 {
 printf("process %d:\n",i+1);
scanf("%d%d",&block[i],&run[i]);
 }
 }
 block[p]=0;
run[p]=0;  for(k=1;
k<=r; k++)
 {
 j=0;  for(i=1;
i<=n; i++)
 {
 totalloc[k]=j+resalloc[i][k]; j=totalloc[k];
}
}
for(i=1; i<=n; i++)
{
if(block[i]==1||run[i]==1)
active[i]=1; else
active[i]=0;
}
 for(k=1; k<=r; k++)
 {
 resalloc[p][k]+=newreq[k];
totalloc[k]+=newreq[k];
 }
```

```
 for(k=1; k<=r; k++)
 {
 if(totext[k]-totalloc[k]<0)
 {
 u=1;
break;
 }
 }
 if(u==0)
 {
 for(k=1; k<=r; k++)
simalloc[k]=totalloc[k];
for(s=1; s<=n; s++)  for(i=1;
i<=n; i++)
 {
 if(active[i]==1)
{
j=0; for(k=1; k<=r;
k++)
{
if((totext[k]-simalloc[k])<(max[i][k]-resalloc[i][k])) {
j=1; break;
}
 }
}
 if(j==0)
 {
 active[i]=0;  for(k=1;
k<=r; k++)
simalloc[k]=resalloc[i][k];
 }
```

```
 }
 m=0;  for(k=1; k<=r; k++)
resreq[p][k]=newreq[k];
printf("Deadlock willn't occur");
 }
 else
 {
 for(k=1; k<=r; k++)
 {
 resalloc[p][k]=newreq[k];
totalloc[k]=newreq[k];
 }
 printf("Deadlock will occur");
}
}
```

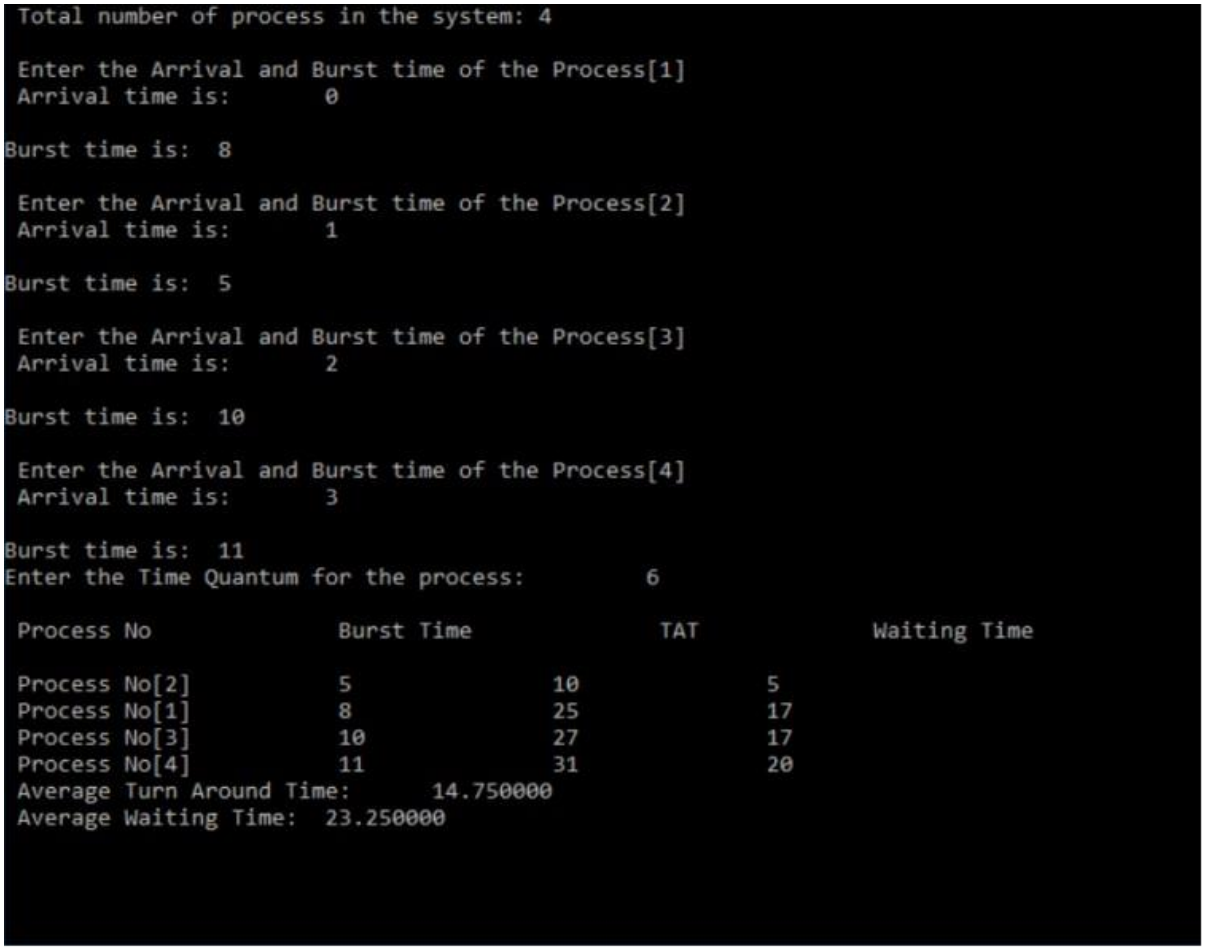


6. Construct a C program to simulate Round Robin scheduling algorithm with C.

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

void 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 %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();
}
```



7. 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);
//creates shared memory segment with key 2345, having size 1024 bytes. IPC_CREAT is used to
create the shared segment if it does not exist. 0666 are the permissions on the shared segment
printf("Key of shared memory is %d\n",shmid);   shared_memory=shmat(shmid,NULL,0);
```

```
//process attached to shared memory segment   printf("Process attached at
%p\n",shared_memory);
//this prints the address where the segment is attached with this process
printf("Enter some data to write to shared memory\n");   read(0,buff,100);
//get some input from user   strcpy(shared_memory,buff); //data written
to shared memory   printf("You wrote : %s\n",(char *)shared_memory);
}
```

```
Key of shared memory is 0
Process attached at 0x7ffe040fb000
Enter some data to write to shared memory
Hello World
You wrote: Hello World
```

8. Illustrate the concept of multithreading using a C program.

```
#include  <stdio.h>  #include
<pthread.h>                  void
*threadFunc(void *arg)
{
        char *str;
int i = 0;
str=(char*)arg;

        while(i < 10 )
        {
                usleep(1);
                printf("threadFunc says: %s\n",str);
                ++i;
        }

        return NULL;
}
```

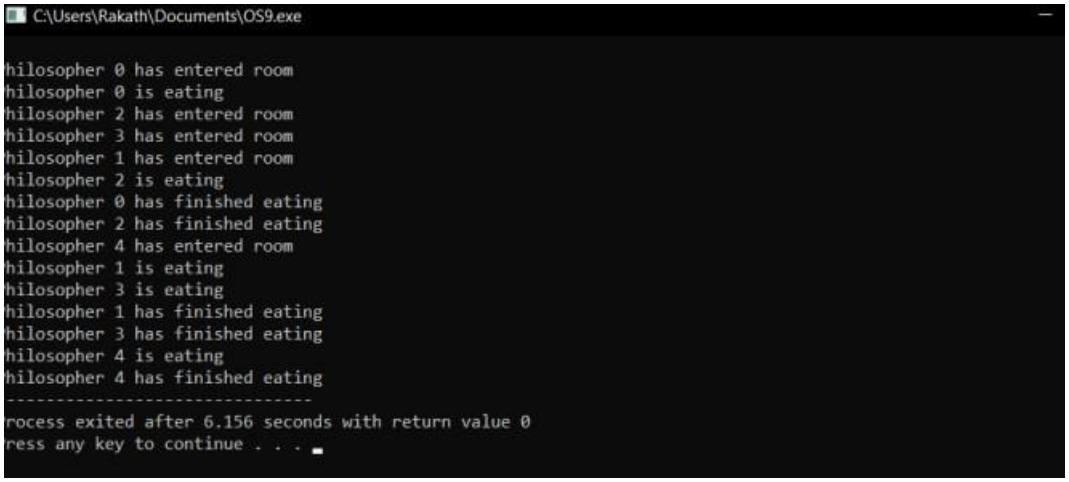
```
int main(void)
{
        pthread_t pth;
int i = 0;

        pthread_create(&pth,NULL,threadFunc,"processing...");

        pthread_join(pth, NULL /* void ** return value could go here */);

        while(i < 10 )
        {
                usleep(1);
                printf("main() is running...\n");
                ++i;
        }

        return 0;
}
```

```
Waiting for the thread to end...
Done. (~2 seconds after)
Thread ended.
```

9. 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()
```

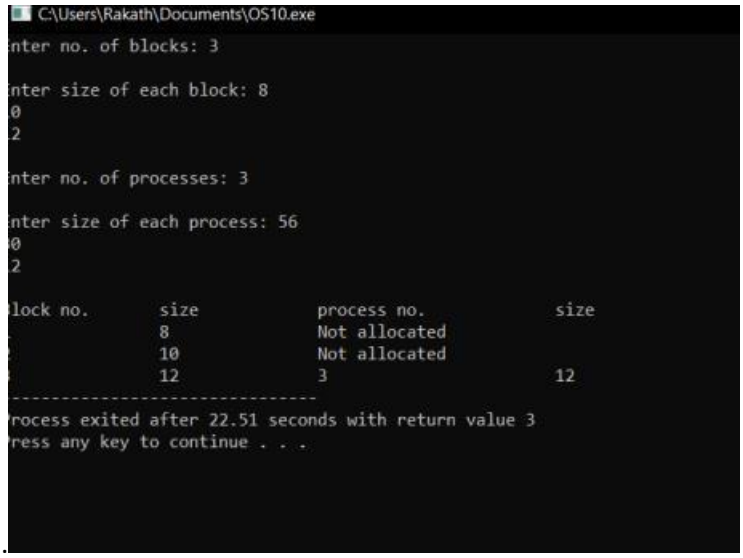
```
{ int
i,a[5];
pthread_t tid[5];
sem_init(&room,0,4); for(i=0;i<5;i++)
sem_init(&chopstick[i],0,1);
for(i=0;i<5;i++){
a[i]=i;
pthread_create(&tid[i],NULL,philosopher,(void *)&a[i]);
}
for(i=0;i<5;i++) pthread_join(tid[i],NULL);
}
void * philosopher(void * num)
{
int phil=*(int *)num;
sem_wait(&room);
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);
}
```



10. Construct a C program for implementation of memory allocation using first fit strategy

```
#include<stdio.h> void main()
{
int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j; for(i
= 0; i < 10; i++)
{ flags[i] =
0;
allocation[
i] = -1;

}
printf("Enter no. of blocks: ");
scanf("%d", &bno); printf("\nEnter size
of each block: "); for(i = 0; i < bno; i++)
scanf("%d", &bsize[i]); printf("\nEnter
no. of processes: "); scanf("%d",
&pno); printf("\nEnter size of each
process: "); for(i = 0; i < pno; i++)
scanf("%d", &psize[i]);
for(i = 0; i < pno; i++)  for(j = 0; j <
bno; j++) if(flags[j] == 0 && bsize[j]
>= psize[i])
```

```
{ allocation[j] =
i; flags[j] = 1;
break;
}
printf("\nBlock no.\tsize\t\tprocess no.\t\tsize"); for(i
= 0; i < bno; i++)
{
printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
if(flags[i] == 1)
printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);
else printf("Not allocated");
}
}
```



.

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

```
#include<stdlib.h>
#include<string.h>
#include<stdio.h> struct
{
char dname[10],fname[10][10];
int fcnt;
```

```
}dir; void
main()
{ int
i,ch;
char f[30]; dir.fcnt = 0; printf("\nEnter
name of directory -- "); scanf("%s",
dir.dname); while(1)
{
printf("\n\n1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Exit\nEnter your choice
-- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\nEnter the name of the file -- ");
scanf("%s",dir.fname[dir.fcnt]); dir.fcnt++;
break; case 2: printf("\nEnter the name of the
file -- "); scanf("%s",f); for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is deleted ",f);
strcpy(dir.fname[i],dir.fname[dir.fcnt-1]); break; } }
if(i==dir.fcnt) printf("File %s not found",f); else
dir.fcnt--;
break; case 3: printf("\nEnter the name of the
file -- "); scanf("%s",f); for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is found ", f); break;
}
```

```
}
if(i==dir.fcnt) printf("File %s
not found",f); break;
case 4: if(dir.fcnt==0)
printf("\nDirectory Empty"); else
{
printf("\nThe Files are -- ");
for(i=0;i<dir.fcnt;i++) printf("\t%s",dir.fname[i]);
}
break; default:
exit(0);
}
}
}
```

```
Output
/tmp/yW7Fsa3gGW.o
Enter name of directory -- student
1. Create File  2. Delete File  3. Search File
 4. Display Files   5. Exit
Enter your choice -- 3
Enter the name of the file -- student
File student not found

1. Create File  2. Delete File  3. Search File
 4. Display Files   5. Exit
Enter your choice -- 5
```

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

```
#include<string.h>
#include<stdlib.h>
#include<stdio.h> struct
{
```

```
char dname[10],fname[10][10];
int       fcnt;
}dir[10];
void main()
{
int i,ch,dcnt,k;
char f[30], d[30];
dcnt=0; while(1)
{
printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
printf("\n4. Search File\t\t5. Display\t6. Exit\tEnter your choice -- ");
scanf("%d",&ch); switch(ch)
{
case 1: printf("\nEnter name of directory -- ");
scanf("%s", dir[dcnt].dname); dir[dcnt].fcnt=0;
dcnt++; printf("Directory created"); break; case 2:
printf("\nEnter name of the directory -- ");
scanf("%s",d); for(i=0;i<dcnt;i++)
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",dir[i].fname[dir[i].fcnt]);
printf("File created"); break;
}
if(i==dcnt)
printf("Directory %s not found",d); break; case 3:
printf("\nEnter name of the directory -- ");
scanf("%s",d); for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
```

```
printf("Enter name of the file -- ");
scanf("%s",f); for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is deleted ",f); dir[i].fcnt--;
strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
goto jmp;
}
}
printf("File %s not found",f); goto
jmp;
}
}
printf("Directory %s not found",d); jmp : break;
case 4: printf("\nEnter name of the directory -- ");
scanf("%s",d); for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter the name of the file -- ");
scanf("%s",f); for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is found ",f);
goto jmp1;
}
}
printf("File %s not found",f); goto
jmp1;
```

```
}
}
printf("Directory %s not found",d);
jmp1: break; case 5: if(dcnt==0)
printf("\nNo Directory's "); else
{
printf("\nDirectory\tFiles"); for(i=0;i<dcnt;i++)
{
printf("\n%s\t\t",dir[i].dname);
for(k=0;k<dir[i].fcnt;k++) printf("\t%s",dir[i].fname[k]);
}
}
break; default:exit(0);
}
}
}
```

```
Output
/tmp/0UpHG6yHpz.o
1. Create Directory 2. Create File  3. Delete File
4. Search File      5. Display  6. Exit Enter your choice -- 1
Enter name of directory -- sss
Directory created

1. Create Directory 2. Create File  3. Delete File
4. Search File      5. Display  6. Exit Enter your choice -- 2
Enter name of the directory -- sss
Enter name of the file -- basha
File created

1. Create Directory 2. Create File  3. Delete File
4. Search F5
ile     5. Display  6. Exit Enter your choice -- 5
Directory   Files
sss

1. Create Directory 2. Create File  3. Delete File
4. Search File      5. Display  6. Exit Enter your choice -- 6
```

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

```
#include<stdio.h>


int main()
{
   FILE *fp;
fp=fopen("scaler.txt","r");
   if(!fp)
   {
      printf("Error: File cannot be opened\n") ;
      return 0;
   }
```

```
   printf("Position pointer in the beginning : %ld\n",ftell(fp));


   char ch;
while(fread(&ch,sizeof(ch),1,fp)==1)
   {
      printf("%c",ch);
   }


   printf("\nSize of file in bytes is : %ld\n",ftell(fp));
fclose(fp);    return 0;
}
```

```
Enter Name : Basha

Enter Age : 24

Enter Salary : 40000

Enter EMP-ID : 19211491

Want to add another record (Y/N) : N
```

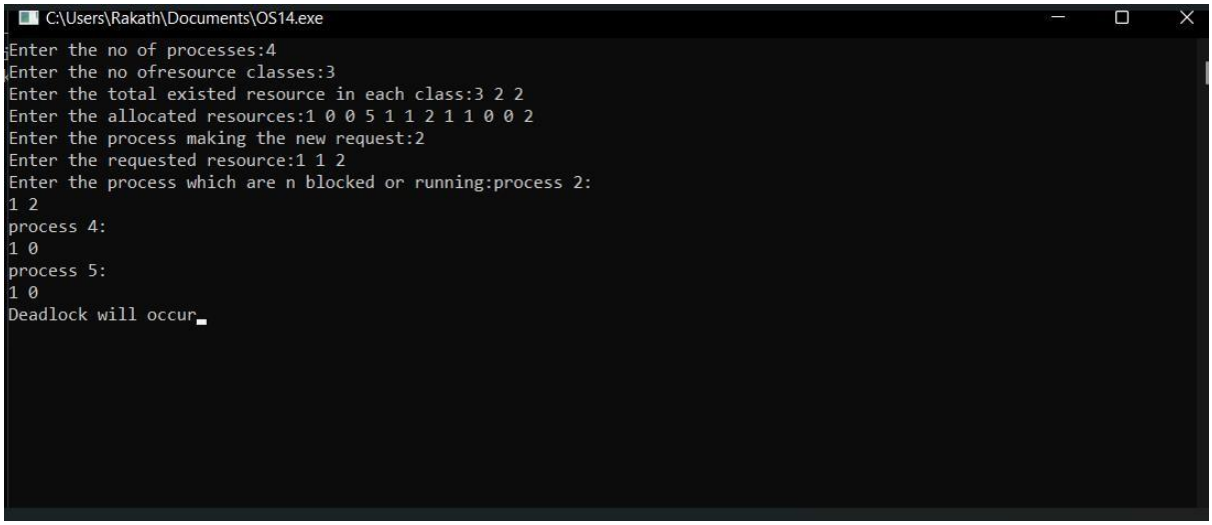
14. Illustrate the deadlock avoidance concept by simulating Banker's algorithm with C.

```
#include<stdio.h>
#include<conio.h> void
main()
{
   int n,r,i,j,k,p,u=0,s=0,m;    int
block[10],run[10],active[10],newreq[10];    int
max[10][10],resalloc[10][10],resreq[10][10];    int
totalloc[10],totext[10],simalloc[10];    //clrscr();

   printf("Enter the no of processes:");    scanf("%d",&n);
printf("Enter the no ofresource classes:");
scanf("%d",&r);    printf("Enter the total existed
resource in each class:");    for(k=1; k<=r; k++)
```

```
scanf("%d",&totext[k]);     printf("Enter the allocated
resources:");     for(i=1; i<=n; i++)        for(k=1; k<=r; k++)
scanf("%d",&resalloc);
   printf("Enter the process making the new request:");
scanf("%d",&p);     printf("Enter the requested resource:");
for(k=1; k<=r; k++)        scanf("%d",&newreq[k]);
printf("Enter the process which are n blocked or running:");
for(i=1; i<=n; i++)
   {
      if(i!=p)
      {
         printf("process %d:\n",i+1);
scanf("%d%d",&block[i],&run[i]);
      }
   }
   block[p]=0;
run[p]=0;     for(k=1;
k<=r; k++)
   {

      j=0;        for(i=1;
i<=n; i++)
      {
         totalloc[k]=j+resalloc[i][k];
j=totalloc[k];
      }
   }
   for(i=1; i<=n; i++)
   {
      if(block[i]==1||run[i]==1)
```

```
          active[i]=1;
else
active[i]=0;
   }
   for(k=1; k<=r; k++)
   {
      resalloc[p][k]+=newreq[k];
totalloc[k]+=newreq[k];
   }
   for(k=1; k<=r; k++)
   {
      if(totext[k]-totalloc[k]<0)
      {
u=1;
break;
      }
   }
   if(u==0)
   {
      for(k=1; k<=r; k++)
simalloc[k]=totalloc[k]; for(s=1; s<=n; s++)
          for(i=1; i<=n; i++)
          {
             if(active[i]==1)
             {                  j=0;
for(k=1; k<=r; k++)
                {
                   if((totext[k]-simalloc[k])<(max[i][k]-resalloc[i][k]))
                   {
j=1;
break;
```

```
                    }
                  }
}
            if(j==0)

            {                   active[i]=0;
for(k=1; k<=r; k++)
simalloc[k]=resalloc[i][k];
            }
          }
      m=0;        for(k=1; k<=r; k++)
resreq[p][k]=newreq[k];
printf("Deadlock willn't occur");
   }
   else
   {
      for(k=1; k<=r; k++)
      {
         resalloc[p][k]=newreq[k];
totalloc[k]=newreq[k];
      }
      printf("Deadlock will occur");
   }
   getch();
}
```



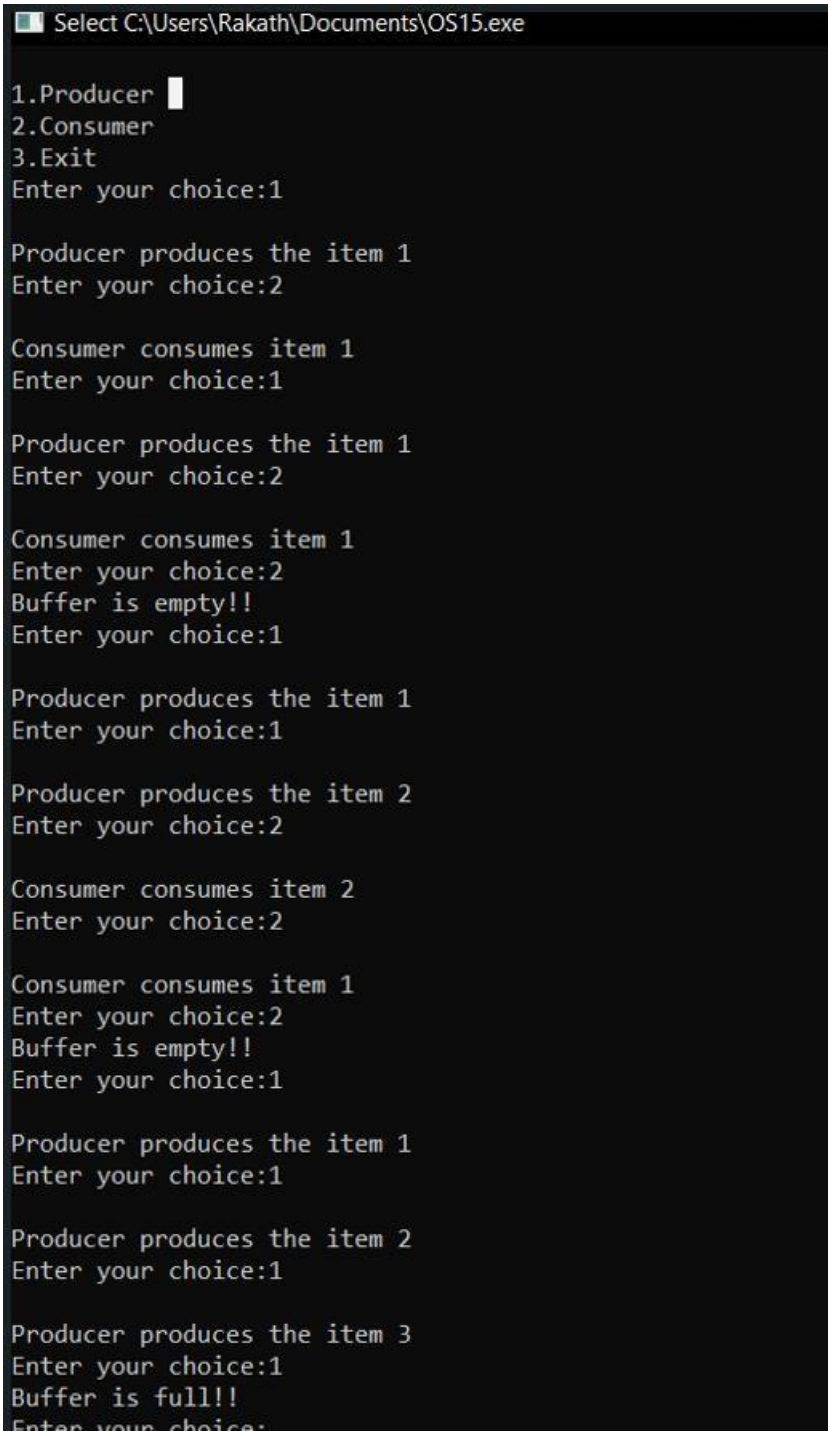
15 Construct a C program to simulate producer-consumer problem using semaphores.

```
#include <stdio.h>

#include <stdlib.h> int

mutex = 1;



int full = 0;



int empty = 10, x = 0;



void producer()

{

   --mutex;



   ++full;



   --empty;
```

```
   x++;
   printf("\nProducer produces"
"item %d",
          x);


   ++mutex;
}


void consumer()
{

   --mutex;


   --full;


   ++empty;     printf("\nConsumer
consumes "
          "item %d",
          x);
x--;

   ++mutex;
}

int main()
```

```
{    int n,
i;
   printf("\n1. Press 1 for Producer"
"\n2. Press 2 for Consumer"
         "\n3. Press 3 for Exit");
#pragma omp critical

   for (i = 1; i > 0; i++) {

      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;
        }
    }
}
```

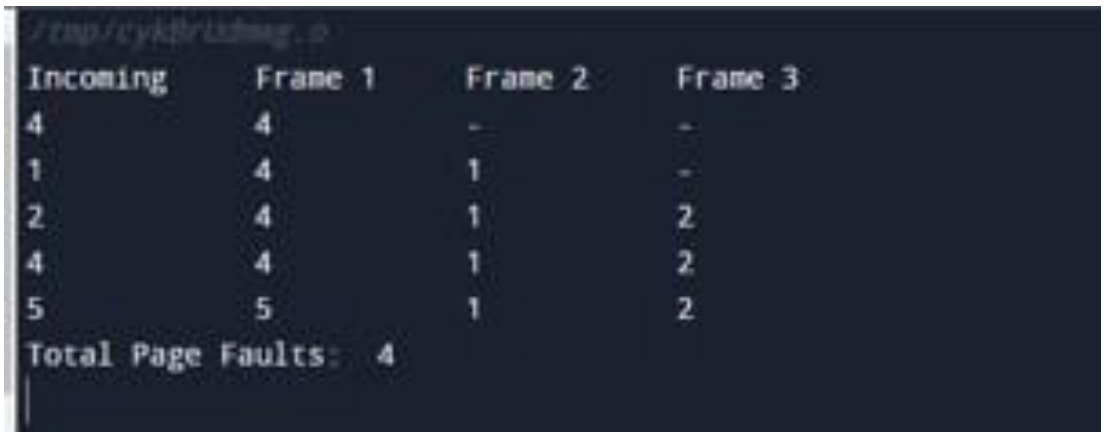


16. Construct a C program to simulate the First in First Out paging technique of memory management.

```
#include <stdio.h> int

main()

{

   int incomingStream[] = {4, 1, 2, 4, 5};
```

```
   int pageFaults = 0;
int frames = 3;     int
m, n, s, pages;

   pages = sizeof(incomingStream)/sizeof(incomingStream[0]);

   printf("Incoming \t Frame 1 \t Frame 2 \t Frame 3");
int temp[frames];     for(m = 0; m < frames; m++)
   {
      temp[m] = -1;
   }

   for(m = 0; m < pages; m++)
   {
      s = 0;

      for(n = 0; n < frames; n++)
      {
         if(incomingStream[m] == temp[n])
         {
s++;
            pageFaults--;
         }
      }
      pageFaults++;

      if((pageFaults <= frames) && (s == 0))
      {
         temp[m] = incomingStream[m];
      }
      else if(s == 0)
```

```
        {
            temp[(pageFaults - 1) % frames] = incomingStream[m];
        }

        printf("\n");
printf("%d\t\t\t",incomingStream[m]);        for(n
= 0; n < frames; n++)
        {
            if(temp[n] != -1)
printf(" %d\t\t\t", temp[n]);
            else
printf(" - \t\t\t");
        }
    }

    printf("\nTotal Page Faults:\t%d\n", pageFaults);
    return 0;
}
```



17. Construct a C program to simulate the Least Recently Used paging technique of memory management.

```
#include <stdio.h>

        #include <limits.h>
```

```
        int checkHit(int incomingPage, int queue[], int occupied){

            for(int i = 0; i < occupied; i++){
if(incomingPage == queue[i])
return 1;
            }

            return 0;
        }

        void printFrame(int queue[], int occupied)
        {
            for(int i = 0; i < occupied; i++)
printf("%d\t\t\t",queue[i]);
        }

        int main()
        {

            int incomingStream[] = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3};
```

```
        int n =
sizeof(incomingStream)/sizeof(incomingStream[0]);      int
frames = 3;          int queue[n];      int distance[n];          int
occupied = 0;        int pagefault = 0;

        printf("Page\t Frame1 \t Frame2 \t Frame3\n");

        for(int i = 0;i < n; i++)
        {
      printf("%d:                              \t\t",incomingStream[i]);
if(checkHit(incomingStream[i],          queue,          occupied)){
printFrame(queue, occupied);
            }       else if(occupied < frames){
       queue[occupied] = incomingStream[i];
       pagefault++;
            occupied++;

            printFrame(queue, occupied);
            }           else{
int max = INT_MIN;          int index;
```

```
                        for (int j = 0; j <
frames; j++)
                {
distance[j] = 0;
                                for(int k = i -
1; k >= 0; k--)
                    {
                       ++distance[j];

                       if(queue[j] == incomingStream[k])
                          break;
                    }

                    if(distance[j] > max){
        max = distance[j];
        index = j;
                    }           }
queue[index] = incomingStream[i];
printFrame(queue, occupied);
pagefault++;
```

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

            printf("Page Fault: %d",pagefault);

            return 0;
        }
```

```
Page      Frame1      Frame2      Frame3
1:          1
2:          1           2
3:          1           2           3
2:          1           2           3
1:          1           2           3
5:          1           2           5
2:          1           2           5
1:          1           2           5
6:          1           2           6
2:          1           2           6
5:          5           2           6
6:          5           2           6
3:          5           3           6
1:          1           3           6
3:          1           3           6
Page Fault: 8
```

18. Construct a C program to simulate the optimal paging technique of memory management

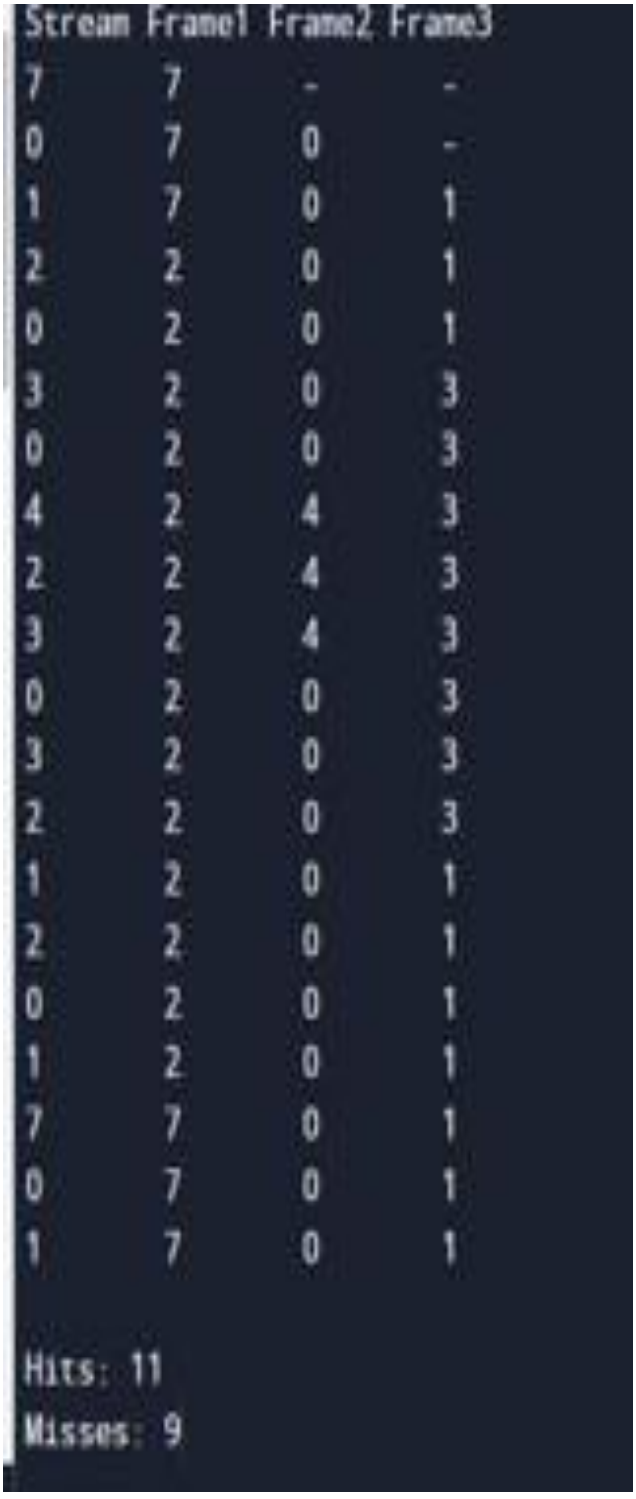
```
#include <stdio.h> int search(int key, int frame_items[],
int frame_occupied)
{
   for (int i = 0; i < frame_occupied; i++)
if (frame_items[i] == key)           return
1;     return 0;
}

void printOuterStructure(int max_frames){
printf("Stream ");

   for(int i = 0; i < max_frames; i++)
printf("Frame%d ", i+1);
}
void printCurrFrames(int item, int frame_items[], int
frame_occupied, int max_frames){     printf("\n%d
\t\t", item);     for(int i = 0; i < max_frames; i++){
if(i < frame_occupied)           printf("%d \t\t",
frame_items[i]);
```

```
        else
printf("- \t\t");
    }
}
int predict(int ref_str[], int frame_items[], int refStrLen, int index, int
frame_occupied)
{
    int result = -1, farthest = index;     for
(int i = 0; i < frame_occupied; i++) {
        int j;
        for (j = index; j < refStrLen; j++)
        {
            if (frame_items[i] == ref_str[j])
            {                if (j >
farthest) {
farthest = j;
result = i;
```

```
            }
break;
          }
       }
       if (j == refStrLen)
return i;
   }
   return (result == -1) ? 0 : result;
}

void optimalPage(int ref_str[], int refStrLen, int frame_items[], int
max_frames)
{
   int frame_occupied = 0;
printOuterStructure(max_frames);

   int hits = 0;     for (int i = 0; i <
refStrLen; i++) {

       if (search(ref_str[i], frame_items, frame_occupied)) {
```

```
            hits++;
            printCurrFrames(ref_str[i], frame_items, frame_occupied,
max_frames);
            continue;
        }

        if (frame_occupied < max_frames){
frame_items[frame_occupied] = ref_str[i];            frame_occupied++;
printCurrFrames(ref_str[i], frame_items, frame_occupied,
max_frames);

        }
else {
            int pos = predict(ref_str, frame_items, refStrLen, i + 1,
frame_occupied);
            frame_items[pos] = ref_str[i];
            printCurrFrames(ref_str[i], frame_items, frame_occupied,
max_frames);
        }

    }
```

```
    printf("\n\nHits: %d\n", hits);
printf("Misses: %d", refStrLen - hits);
}
int main()
{     int ref_str[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0,
1};     int refStrLen = sizeof(ref_str) / sizeof(ref_str[0]);
    int max_frames = 3;     int
frame_items[max_frames];

    optimalPage(ref_str, refStrLen, frame_items, max_frames);
    return 0;
}
```

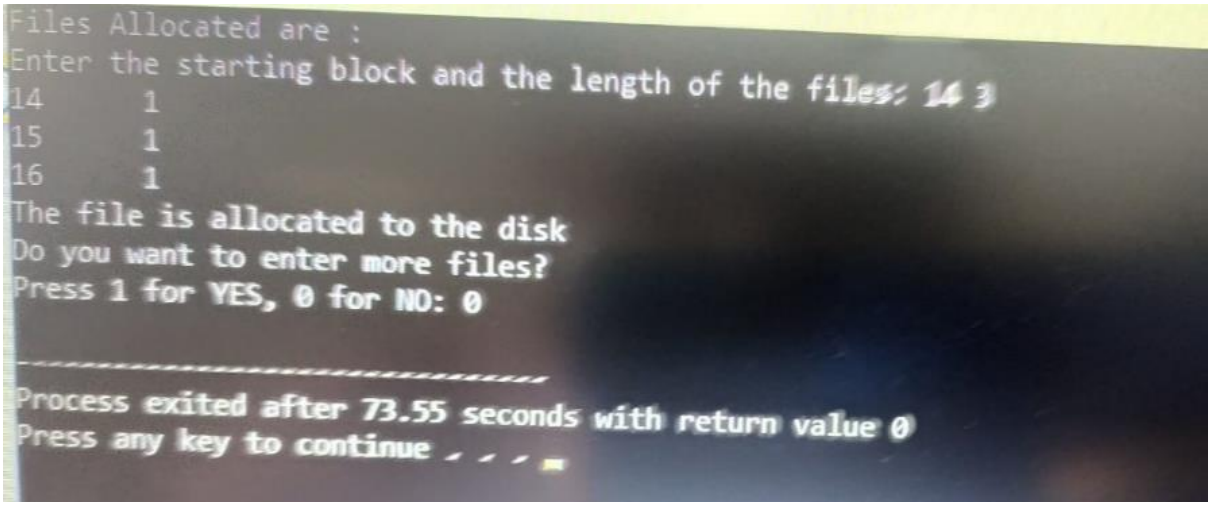


19. Consider a file system where the records of the file are stored one after another both physically and logically. A record of the file can only be accessed by reading all the previous records. Design a C program to simulate the file allocation strategy.

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

```
#include <conio.h> #include <stdlib.h> void recurse(int files[]){
int flag = 0, startBlock, len, j, k, ch;     printf("Enter the starting
block and the length of the files: ");     scanf("%d%d",
&startBlock, &len);     for (j=startBlock; j<(startBlock+len); j++){
       if (files[j] == 0)
flag++;
   }
   if(len == flag){        for (int k=startBlock;
k<(startBlock+len); k++){
          if (files[k] == 0){               files[k]
= 1;               printf("%d\t%d\n", k,
files[k]);
          }
       }
       if (k != (startBlock+len-1))           printf("The
file is allocated to the disk\n"); }
   else printf("The file is not allocated to the
disk\n");     printf("Do you want to enter more
files?\n");     printf("Press 1 for YES, 0 for NO: ");
```

```
scanf("%d", &ch);     if (ch == 1)        recurse(files);

else        exit(0);     return;

}

int main()

{ int files[50]; for(int

i=0;i<50;i++) files[i]=0;

printf("Files Allocated are :\n");

recurse(files);


getch(); return

0;

}
```

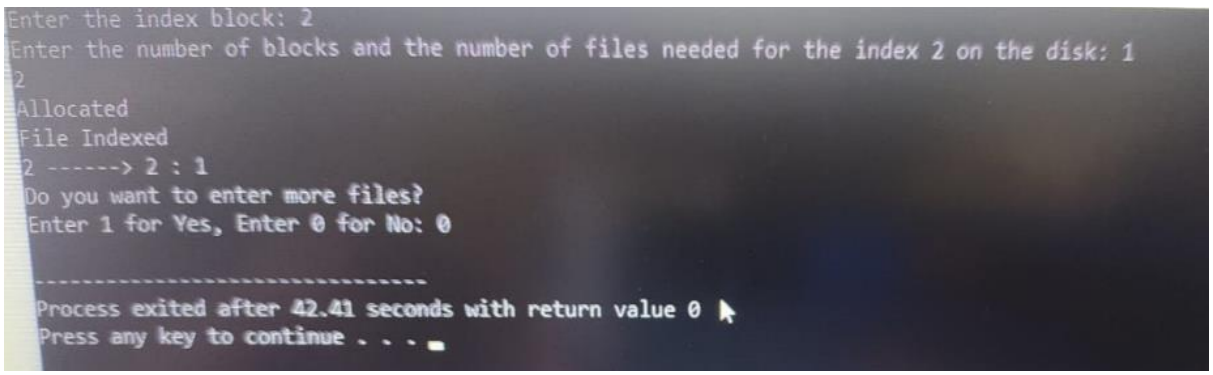


20. Consider a file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a C program to simulate the file allocation strategy.

```
#include <stdio.h>

#include <conio.h> #include <stdlib.h>

int files[50], indexBlock[50], indBlock,

n; void recurse1(); void recurse2(); void

recurse1(){     printf("Enter the index

block: "); scanf("%d", &indBlock); if

(files[indBlock] != 1){

       printf("Enter the number of blocks and the number of files needed for the index %d on the disk: ", indBlock);

scanf("%d", &n);

   }

   else{        printf("%d is already allocated\n",

indBlock);        recurse1();

   }

   recurse2();

}

void recurse2(){

   int ch;     int flag = 0;     for (int

i=0; i<n; i++){        scanf("%d",
```

```
&indexBlock[i]);        if
(files[indexBlock[i]] == 0)
flag++;

   } if (flag == n){ for (int j=0;
j<n; j++){
files[indexBlock[j]] = 1;
       }
       printf("Allocated\n");
printf("File Indexed\n");        for
(int k=0; k<n; k++){
          printf("%d ------> %d : %d\n", indBlock, indexBlock[k],
files[indexBlock[k]]);
       }
   }
   else{        printf("File in the index is already
allocated\n");        printf("Enter another indexed
file\n");        recurse2();
   }
```

```
   printf("Do you want to enter more files?\n");
printf("Enter 1 for Yes, Enter 0 for No: "); scanf("%d", &ch);
if (ch == 1) recurse1();
   else
exit(0);
return;
}
int main()
{
   for(int i=0;i<50;i++)
files[i]=0;
recurse1();     return
0;
}
```



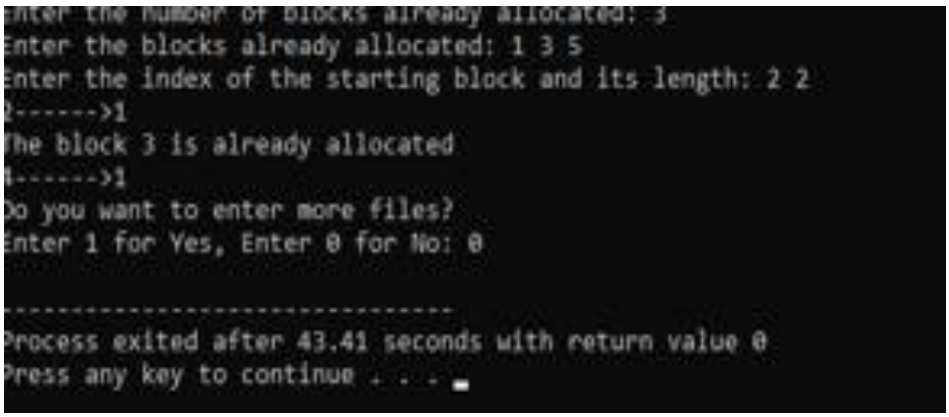
21. With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file.

Each block contains a pointer to the next block. Design a C program to simulate the file allocation strategy.

```
#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

void recursivePart(int pages[]){

   int st, len, k, c, j;

   printf("Enter the index of the starting block and its length: ");

   scanf("%d%d", &st, &len);

   k = len;

   if (pages[st] == 0){

      for (j = st; j < (st + k); j++){

         if (pages[j] == 0){

            pages[j] = 1;

            printf("%d------>%d\n", j, pages[j]);

         }

         else {

            printf("The block %d is already allocated \n", j);

            k++;

         }
```

```
        }
    }
    else
        printf("The block %d is already allocated \n", st);
    printf("Do you want to enter more files? \n");
    printf("Enter 1 for Yes, Enter 0 for No: ");
    scanf("%d", &c);
    if (c==1)
        recursivePart(pages);
    else
        exit(0);
    return;
}
int main(){
    int pages[50], p, a;
    for (int i = 0; i < 50; i++)
        pages[i] = 0;
    printf("Enter the number of blocks already allocated: ");
    scanf("%d", &p);
```

```
    printf("Enter the blocks already allocated: ");
    for (int i = 0; i < p; i++){
        scanf("%d", &a);
        pages[a] = 1;
    }
    recursivePart(pages);
    getch();
    return 0;
}
```

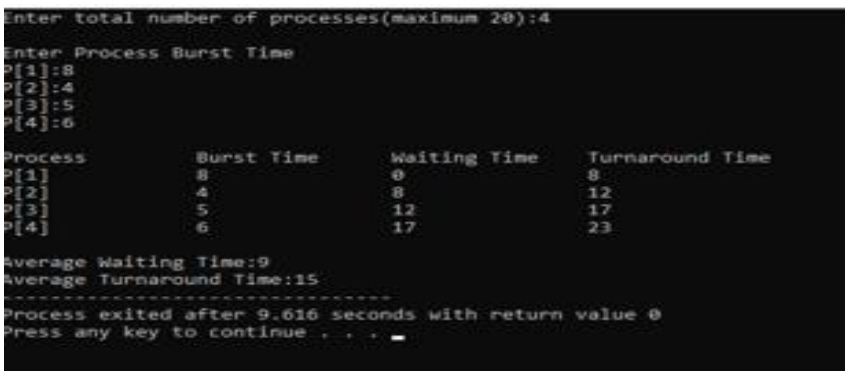


22. Construct a C program to simulate the First Come First Served disk scheduling algorithm.

```
#include<stdio.h>
int main()
{
    int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
    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:%d",avwt);
    printf("\nAverage Turnaround Time:%d",avtat);

    return 0;
}
```



23. Design a C program to simulate SCAN disk scheduling algorithm.

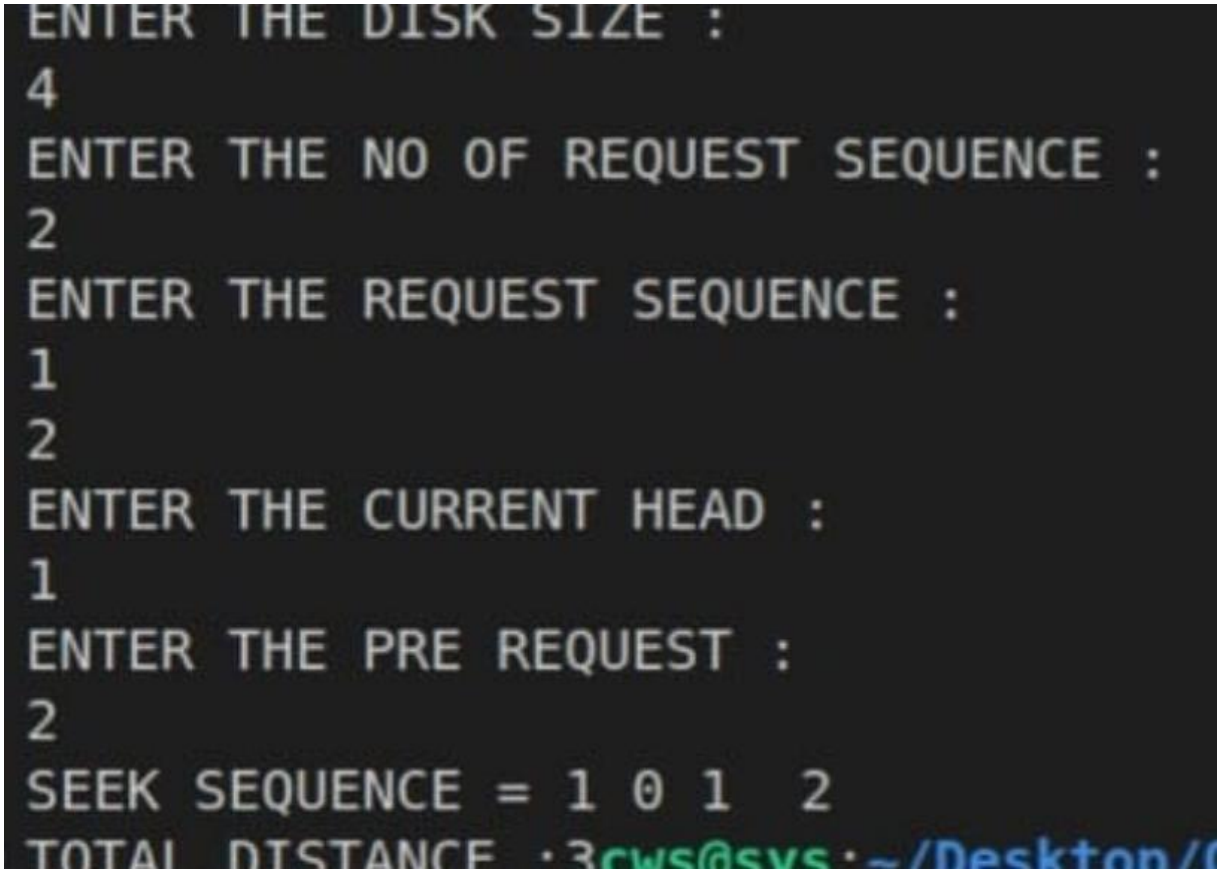
```
#include <stdio.h>
int request[50];
int SIZE;
int pre;
int head;
int uptrack;
int downtrack;
struct max{
   int up;
   int down;
} kate[50];
int dist(int a, int b){
   if (a > b)
      return a - b;
   return b - a;
}
void sort(int n){
   int i, j;
```

```
for (i = 0; i < n - 1; i++){
   for (j = 0; j < n - i - 1; j++){
      if (request[j] > request[j + 1]){
         int temp = request[j];
         request[j] = request[j + 1];
         request[j + 1] = temp;
      }
   }
}
j = 0;
i = 0;
while (request[i] != head){
   kate[j].down = request[i];
   j++;
   i++;
}
downtrack = j;
i++;
j = 0;
```

```
    while (i < n){
        kate[j].up = request[i];
        j++;
        i++;
    }
    uptrack = j;
}
void scan(int n){
    int i;
    int seekcount = 0;
    printf("SEEK SEQUENCE = ");
    sort(n);
    if (pre < head){
        for (i = 0; i < uptrack; i++){
            printf("%d ", head);
            seekcount = seekcount + dist(head, kate[i].up);
            head = kate[i].up;
        }
        for (i = downtrack - 1; i > 0; i--){
```

```
            printf("%d ", head);
            seekcount = seekcount + dist(head, kate[i].down);
            head = kate[i].down;
        }
    }
    else{
        for (i = downtrack - 1; i >= 0; i--){
            printf("%d ", head);
            seekcount = seekcount + dist(head, kate[i].down);
            head = kate[i].down;
        }
        for (i = 0; i < uptrack - 1; i++){
            printf("%d ", head);
            seekcount = seekcount + dist(head, kate[i].up);
            head = kate[i].up;
        }
    }
    printf(" %d\nTOTAL DISTANCE :%d", head, seekcount);
}
```

```
int main(){
    int n, i;
    printf("ENTER THE DISK SIZE :\n");
    scanf("%d", &SIZE);
    printf("ENTER THE NO OF REQUEST SEQUENCE :\n");
    scanf("%d", &n);
    printf("ENTER THE REQUEST SEQUENCE :\n");
    for (i = 0; i < n; i++)
        scanf("%d", &request[i]);
    printf("ENTER THE CURRENT HEAD :\n");
    scanf("%d", &head);
    request[n] = head;
    request[n + 1] = SIZE - 1;
    request[n + 2] = 0;
    printf("ENTER THE PRE REQUEST :\n");
    scanf("%d", &pre);
    scan(n + 3);
}
```



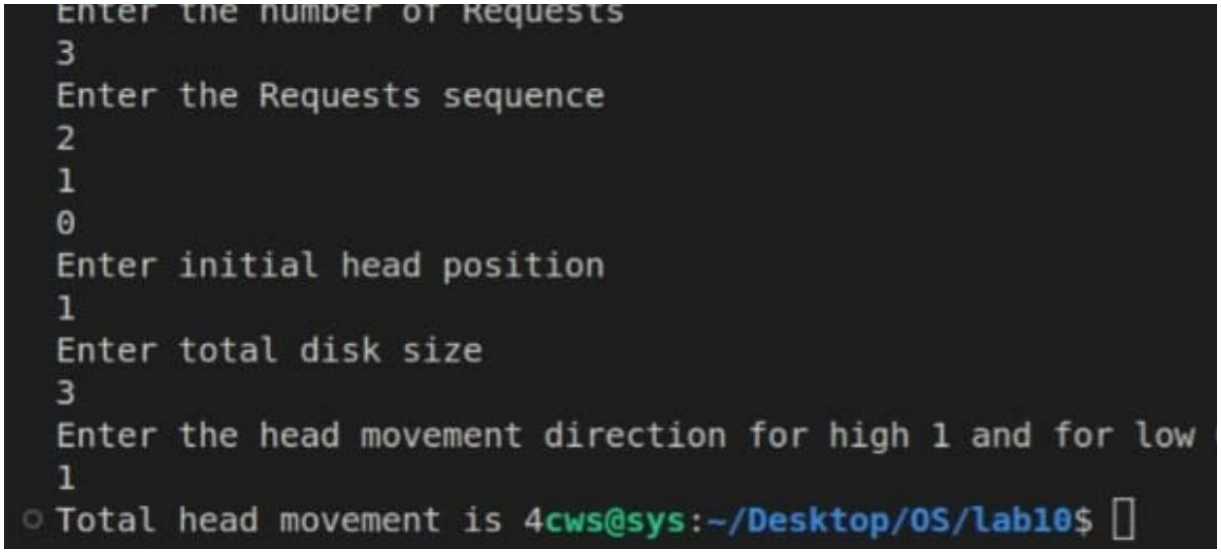
24.. Develop a C program to simulate C-SCAN disk scheduling algorithm.

```
#include <stdio.h>

#include <stdlib.h>

int main(){

   int RQ[100], i, j, n, TotalHeadMoment = 0, initial, size, move;

   printf("Enter the number of Requests\n");

   scanf("%d", &n);

   printf("Enter the Requests sequence\n");

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

```
    scanf("%d", &RQ[i]);
  printf("Enter initial head position\n");
  scanf("%d", &initial);
  printf("Enter total disk size\n");
  scanf("%d", &size);
  printf("Enter the head movement direction for high 1 and for low
0\n");
  scanf("%d", &move);
  for (i = 0; i < n; i++){
    for (j = 0; j < n - i - 1; j++){
      if (RQ[j] > RQ[j + 1]){
        int temp;
        temp = RQ[j];
        RQ[j] = RQ[j + 1];
        RQ[j + 1] = temp;
      }
    }
  }
  int index;
```

```
for (i = 0; i < n; i++){
    if (initial < RQ[i]){
        index = i;
        break;
    }
}
if (move == 1){
    for (i = index; i < n; i++){
        TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
        initial = RQ[i];
    }
    TotalHeadMoment = TotalHeadMoment + abs(size - RQ[i - 1] - 1);
    TotalHeadMoment = TotalHeadMoment + abs(size - 1 - 0);
    initial = 0;
    for (i = 0; i < index; i++){
        TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
        initial = RQ[i];
    }
}
```

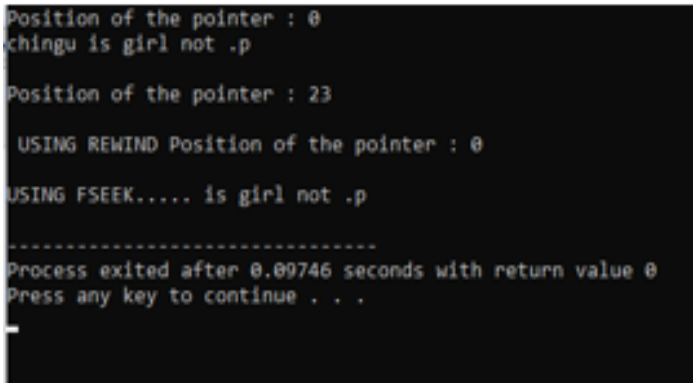
```
    else{
        for (i = index - 1; i >= 0; i--){
            TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
            initial = RQ[i];
        }
        TotalHeadMoment = TotalHeadMoment + abs(RQ[i + 1] - 0);
        TotalHeadMoment = TotalHeadMoment + abs(size - 1 - 0);
        initial = size - 1;
        for (i = n - 1; i >= index; i--){
            TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
            initial = RQ[i];
        }
    }
    printf("Total head movement is %d", TotalHeadMoment);
    return 0;
}
```



25. Illustrate the various File Access Permission and different types users in Linux.

```
#include<stdio.h>

int main()

{

   FILE *fp;

   fp = fopen("C:\\Users\\DELL\\Documents\\file1.txt","r");

   if(!fp)

   {

      printf("Error in opening file\n");

      return 0;

   }

   //Initially the file pointer points to the starting of the file.
```

```
printf("Position of the pointer : %ld\n",ftell(fp));

char ch;

while(fread(&ch,sizeof(ch),1,fp)==1)

{

   //Here we traverse the entire file and print it's contents until we reach it's end.

   printf("%c",ch);

}

printf("\nPosition of the pointer : %ld\n",ftell(fp));


//Below rewind() is going to bring it back to it's original position.

rewind(fp);

printf("\n USING REWIND Position of the pointer : %ld\n",ftell(fp)):

printf("\nUSING FSEEK.....");

fseek(fp, 6, 0);

while(fread(&ch,sizeof(ch),1,fp)==1)

{
```


Position of the pointer : 0
chingu is girl not .p
Position of the pointer : 23
USING REWIND Position of the pointer : 0
USING FSEEK..... is girl not .p

Process exited after 0.09746 seconds with return value 0
Press any key to continue . . .