Operating system

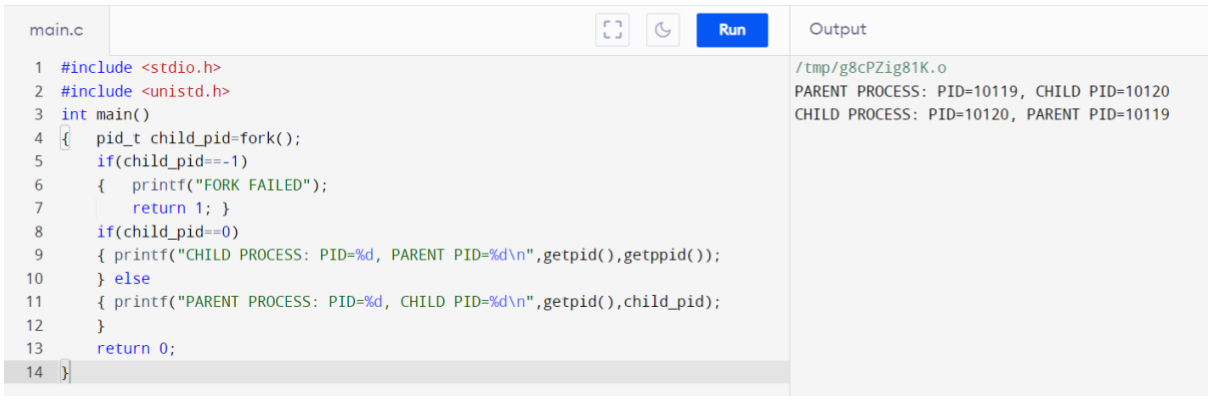
Name: Anshul jolly joshi **Regno: 192211973**

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(){
Pid_t child_pid=fork();
If (child_pid == -1){
Perror("fork failed);
Return 1;
}
else{
printf(" parent Process : PID= %d, child PID=%d\n", getpid() ),child_pid);}
 return 0;
}
```

INPUT AND OUTPUT:



```
#include <stdio.h>
#include <unistd.h>
int main()
{   pid_t child_pid=fork();
    if(child_pid==-1)
    {   printf("FORK FAILED");
        return 1; }
    if(child_pid==0)
    { printf("CHILD PROCESS: PID=%d, PARENT PID=%d\n",getpid(),getppid());
    } else
    { printf("PARENT PROCESS: PID=%d, CHILD PID=%d\n",getpid(),child_pid);
    }
    return 0;
}
```

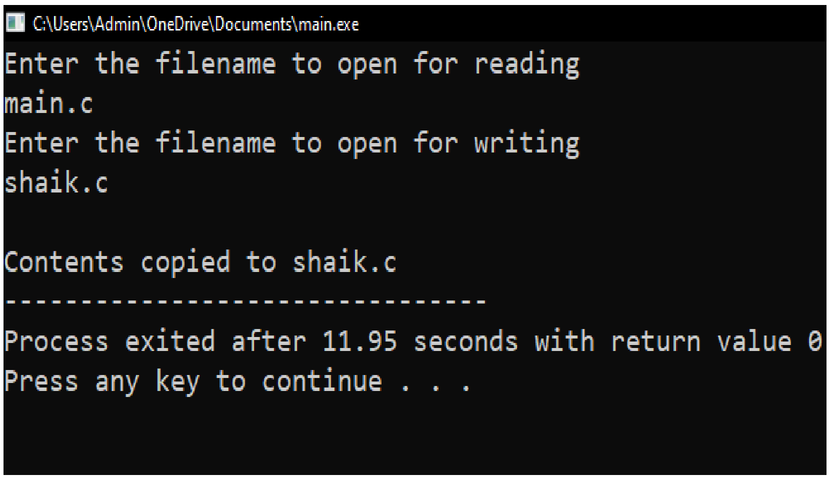
```
/tmp/g8cPZig81K.o
PARENT PROCESS: PID=10119, CHILD PID=10120
CHILD PROCESS: PID=10120, PARENT PID=10119
```

2.To 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 *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;
}
```

INPUT AND OUTPUT:

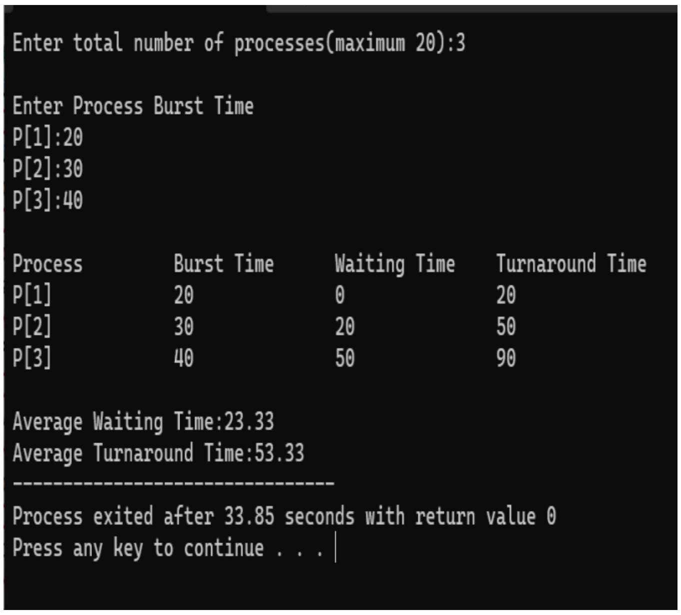


```
C:\Users\Admin\OneDrive\Documents\main.exe
Enter the filename to open for reading
main.c
Enter the filename to open for writing
shaik.c

Contents copied to shaik.c
--------------------------------
Process exited after 11.95 seconds with return value 0
Press any key to continue . . .
```

3.ToDesign 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>
void 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("\nAverage Turnaround Time:%.2f",avtat);
}
```

INPUT AND OUTPUT:



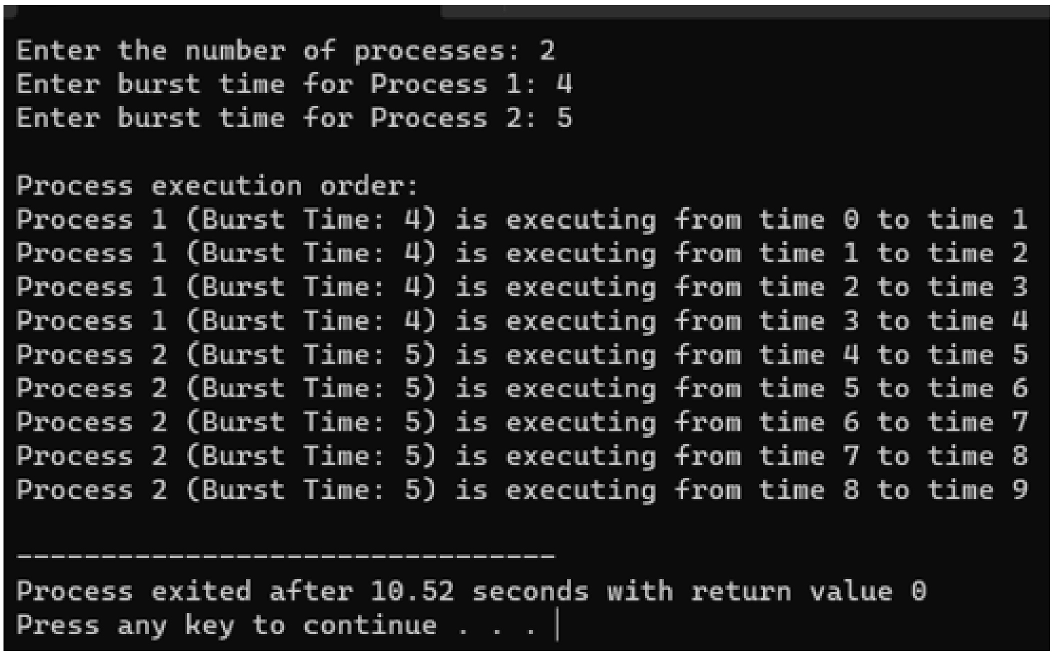
```
Enter total number of processes(maximum 20):3

Enter Process Burst Time
P[1]:20
P[2]:30
P[3]:40

Process         Burst Time      Waiting Time    Turnaround Time
P[1]            20              0               20
P[2]            30              20              50
P[3]            40              50              90

Average Waiting Time:23.33
Average Turnaround Time:53.33
--------------------------------
Process exited after 33.85 seconds with return value 0
Press any key to continue . . .
```

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

PROGRAM:

```
#include <stdio.h>
struct Process {
   int id;
   int burst_time;
};
int main() {
   int n;
   printf("Enter the number of processes: ");
   scanf("%d", &n);

   struct Process processes[n];
   int remaining_time[n];
   for (int i = 0; i < n; i++) {
      processes[i].id = i + 1;
      printf("Enter burst time for Process %d: ", i + 1);
      scanf("%d", &processes[i].burst_time);
      remaining_time[i] = processes[i].burst_time;
   }
   int current_time = 0;
   printf("\nProcess execution order:\n");
   while (1) {
      int smallest = -1;
      for (int i = 0; i < n; i++) {
         if (remaining_time[i] > 0) {
            if (smallest == -1 || remaining_time[i] < remaining_time[smallest]) {
               smallest = i;
            }
         }
      }
      if (smallest == -1) {
         break;
      }
      printf("Process %d (Burst Time: %d) is executing from time %d to ", processes[smallest].id,
processes[smallest].burst_time, current_time);
      current_time += 1;
      remaining_time[smallest] -= 1;
      printf("time %d\n", current_time);
   }
   return 0;
}
```

OUTPUT:



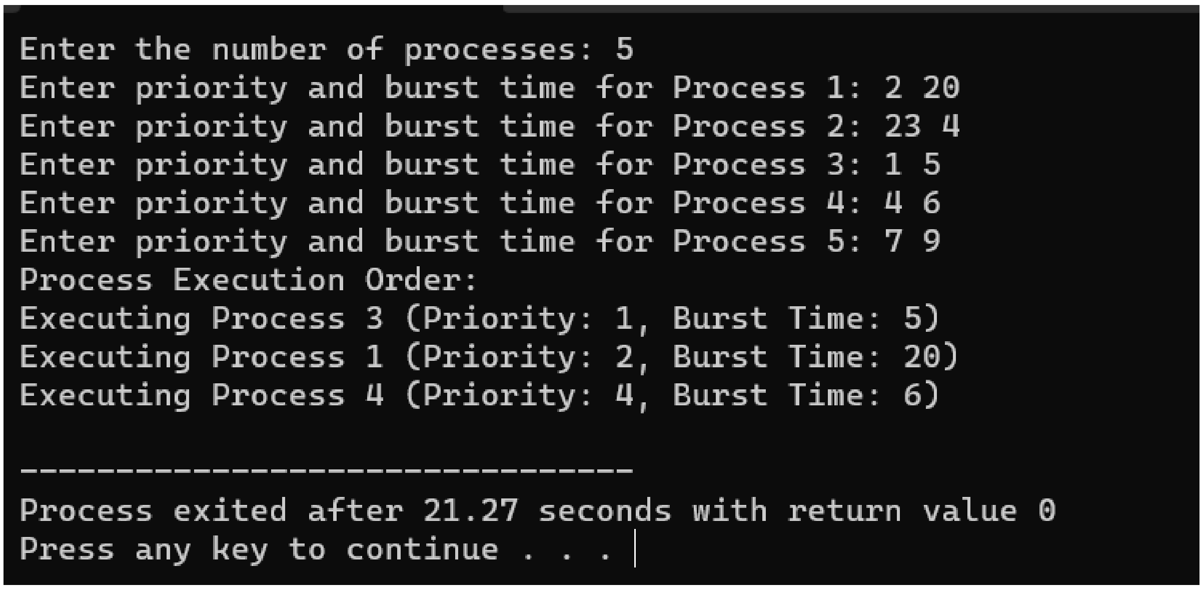
```
Enter the number of processes: 2
Enter burst time for Process 1: 4
Enter burst time for Process 2: 5

Process execution order:
Process 1 (Burst Time: 4) is executing from time 0 to time 1
Process 1 (Burst Time: 4) is executing from time 1 to time 2
Process 1 (Burst Time: 4) is executing from time 2 to time 3
Process 1 (Burst Time: 4) is executing from time 3 to time 4
Process 2 (Burst Time: 5) is executing from time 4 to time 5
Process 2 (Burst Time: 5) is executing from time 5 to time 6
Process 2 (Burst Time: 5) is executing from time 6 to time 7
Process 2 (Burst Time: 5) is executing from time 7 to time 8
Process 2 (Burst Time: 5) is executing from time 8 to time 9

--------------------------------
Process exited after 10.52 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 Process {
   int id;
   int priority;
   int burst_time;
};

int main() {
   int n;
   printf("Enter the number of processes: ");
   scanf("%d", &n);

   struct Process processes[n];

   for (int i = 0; i < n; i++) {
      processes[i].id = i + 1;
      printf("Enter priority and burst time for Process %d: ", i + 1);
      scanf("%d %d", &processes[i].priority, &processes[i].burst_time);
   }
```

```
  printf("Process Execution Order:\n");

  for (int i = 0; i < n; i++) {
    int highest_priority_idx = 0;
    for (int j = 1; j < n; j++) {
      if (processes[j].priority < processes[highest_priority_idx].priority) {
        highest_priority_idx = j;
      }
    }

    printf("Executing Process %d (Priority: %d, Burst Time: %d)\n",
        processes[highest_priority_idx].id, processes[highest_priority_idx].priority,
processes[highest_priority_idx].burst_time);

    for (int j = highest_priority_idx; j < n - 1; j++) {
      processes[j] = processes[j + 1];
    }
    n--;
  }

  return 0;
}
```
OUTPUT:



```
Enter the number of processes: 5
Enter priority and burst time for Process 1: 2 20
Enter priority and burst time for Process 2: 23 4
Enter priority and burst time for Process 3: 1 5
Enter priority and burst time for Process 4: 4 6
Enter priority and burst time for Process 5: 7 9
Process Execution Order:
Executing Process 3 (Priority: 1, Burst Time: 5)
Executing Process 1 (Priority: 2, Burst Time: 20)
Executing Process 4 (Priority: 4, Burst Time: 6)

--------------------------------
Process exited after 21.27 seconds with return value 0
Press any key to continue . . .
```

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

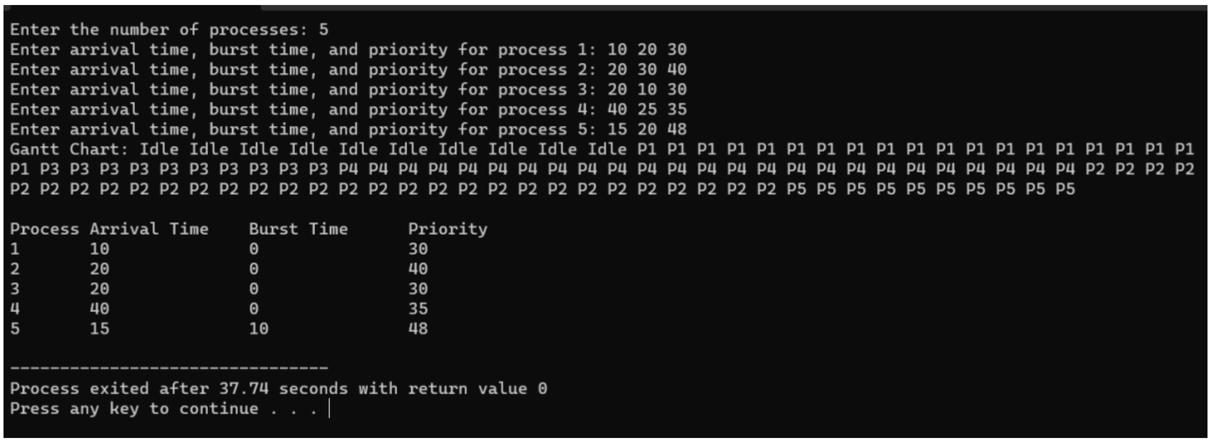
PROGRAM:

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

```
struct Process {
   int id, arrival_time, burst_time, priority;
};

int main() {
   int n;
   printf("Enter the number of processes: ");
   scanf("%d", &n);

   struct Process processes[n];

   for (int i = 0; i < n; i++) {
      processes[i].id = i + 1;
      printf("Enter arrival time, burst time, and priority for process %d: ", processes[i].id);
      scanf("%d %d %d", &processes[i].arrival_time, &processes[i].burst_time, &processes[i].priority);
   }

   int current_time = 0;
   int total_time = 0;

   for (int i = 0; i < n; i++) {
      total_time += processes[i].burst_time;
   }

   printf("Gantt Chart: ");

   while (current_time < total_time) {
      int highest_priority = 9999; // A high value to represent the lowest priority
      int selected_process = -1;

      for (int i = 0; i < n; i++) {
         if (processes[i].arrival_time <= current_time && processes[i].burst_time > 0 && processes[i].priority <
highest_priority) {
            highest_priority = processes[i].priority;
            selected_process = i;
         }
      }

      if (selected_process == -1) {
         printf("Idle ");
         current_time++;
      } else {
         printf("P%d ", processes[selected_process].id);
         processes[selected_process].burst_time--;
         current_time++;
      }
   }
```

```
    printf("\n\nProcess\tArrival Time\tBurst Time\tPriority\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].arrival_time, processes[i].burst_time,
processes[i].priority);
    }

    return 0;
}
```

OUTPUT:



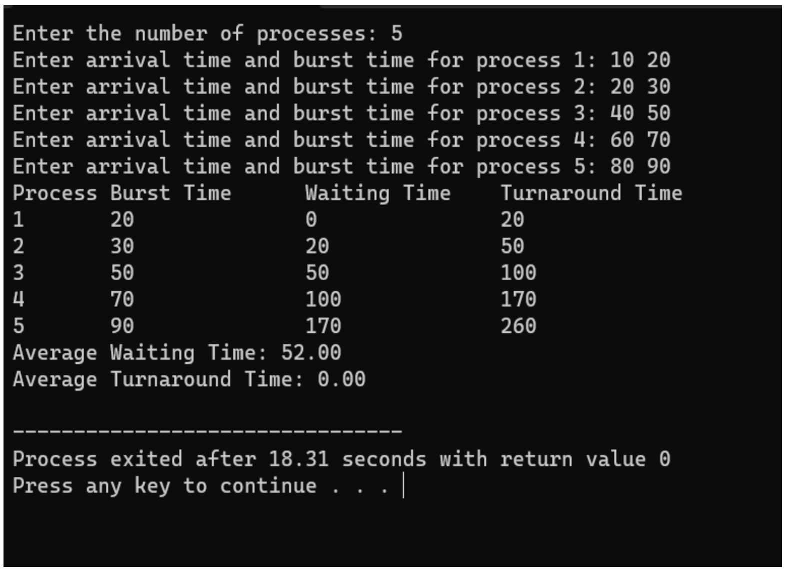
```
Enter the number of processes: 5
Enter arrival time, burst time, and priority for process 1: 10 20 30
Enter arrival time, burst time, and priority for process 2: 20 30 40
Enter arrival time, burst time, and priority for process 3: 20 10 30
Enter arrival time, burst time, and priority for process 4: 40 25 35
Enter arrival time, burst time, and priority for process 5: 15 20 48
Gantt Chart: Idle Idle Idle Idle Idle Idle Idle Idle Idle Idle P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1 P1
P1 P3 P3 P3 P3 P3 P3 P3 P3 P3 P3 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P4 P2 P2 P2 P2
P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5

Process Arrival Time    Burst Time      Priority
1       10              0               30
2       20              0               40
3       20              0               30
4       40              0               35
5       15              10              48

--------------------------------
Process exited after 37.74 seconds with return value 0
Press any key to continue . . .
```

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

PROGRAM:

```
#include <stdio.h>

struct Process {
    int id, arrival_time, burst_time;
};

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process processes[n];

    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        printf("Enter arrival time and burst time for process %d: ", processes[i].id);
        scanf("%d %d", &processes[i].arrival_time, &processes[i].burst_time);
    }
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (processes[i].burst_time > processes[j].burst_time) {
                struct Process temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
            }
        }
    }

    int waiting_time = 0, turnaround_time = 0;
    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

    for (int i = 0; i < n; i++) {
        turnaround_time += processes[i].burst_time;
        printf("%d\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].burst_time,
waiting_time, turnaround_time);
        waiting_time += processes[i].burst_time;
    }

    float avg_waiting_time = (float)waiting_time / n;
    float avg_turnaround_time = (float)turnaround_time / n;

    printf("Average Waiting Time: %.2f\n", avg_waiting_time);
    printf("Average Turnaround Time: %.2f\n");

    return 0;
}
```

OUTPUT:

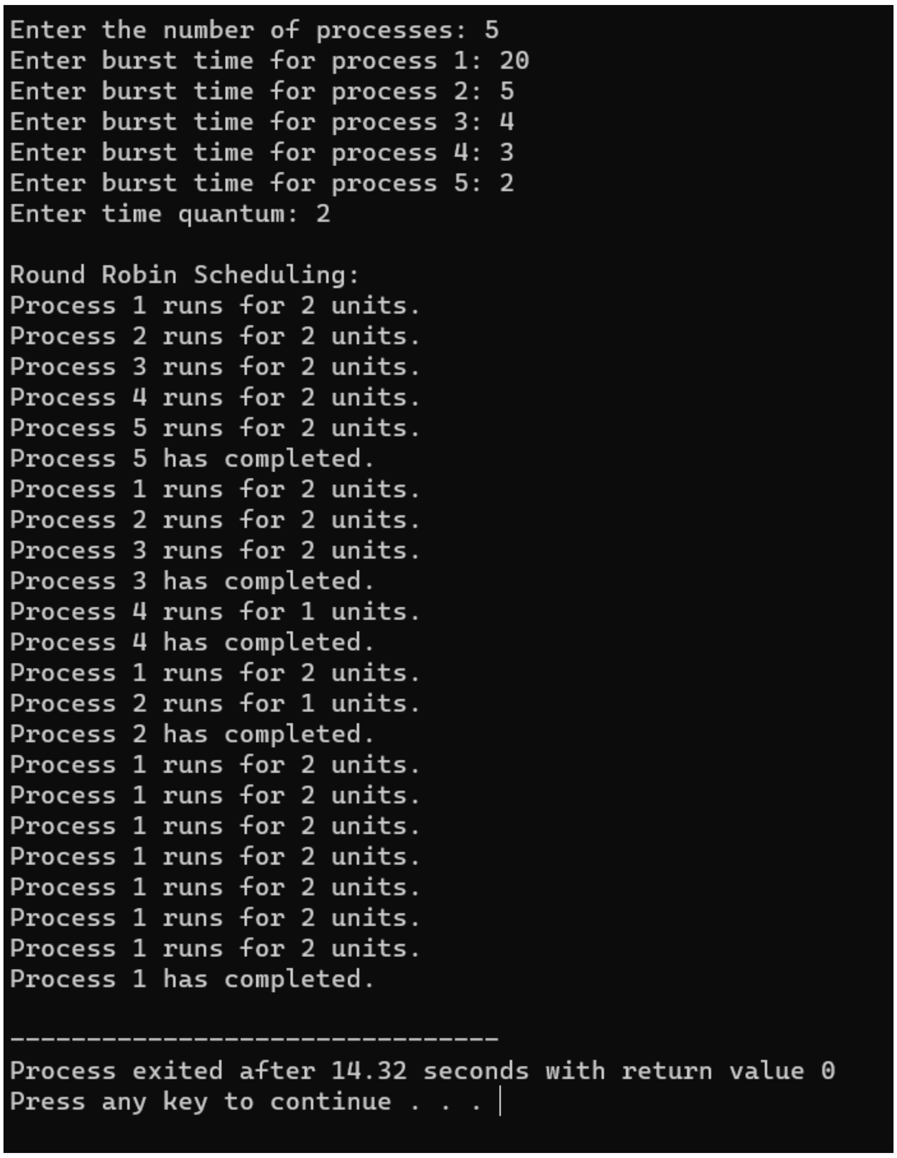


```
Enter the number of processes: 5
Enter arrival time and burst time for process 1: 10 20
Enter arrival time and burst time for process 2: 20 30
Enter arrival time and burst time for process 3: 40 50
Enter arrival time and burst time for process 4: 60 70
Enter arrival time and burst time for process 5: 80 90
Process Burst Time      Waiting Time    Turnaround Time
1       20              0               20
2       30              20              50
3       50              50              100
4       70              100             170
5       90              170             260
Average Waiting Time: 52.00
Average Turnaround Time: 0.00

--------------------------------
Process exited after 18.31 seconds with return value 0
Press any key to continue . . .
```

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

PROGRAM:

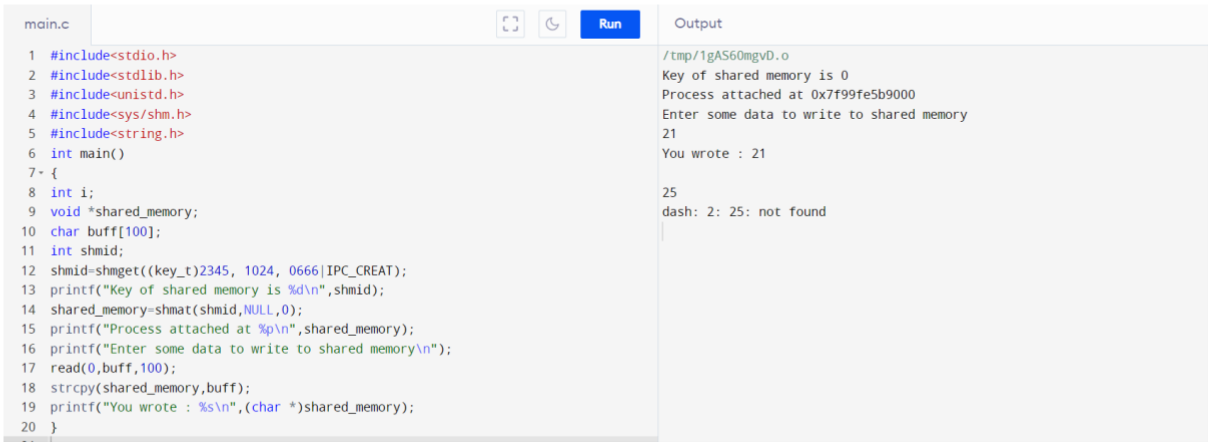
```
#include <stdio.h>

struct Process {
    int id, burstTime, remainingTime;
};
void roundRobin(struct Process processes[], int n, int quantum) {
    int time = 0, completed = 0;
    while (completed < n) {
        for (int i = 0; i < n; i++) {
            if (processes[i].remainingTime > 0) {
                int execTime = (processes[i].remainingTime < quantum) ?
processes[i].remainingTime : quantum;
                processes[i].remainingTime -= execTime;
                time += execTime;
                printf("Process %d runs for %d units.\n", processes[i].id, execTime);
                if (processes[i].remainingTime == 0) {
                    completed++;
                    printf("Process %d has completed.\n", processes[i].id);
                }
            }
        }
    }
}
int main() {
    int n, quantum;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct Process processes[n];
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        printf("Enter burst time for process %d: ", processes[i].id);
        scanf("%d", &processes[i].burstTime);
        processes[i].remainingTime = processes[i].burstTime;
    }
    printf("Enter time quantum: ");
    scanf("%d", &quantum);
    printf("\nRound Robin Scheduling:\n");
    roundRobin(processes, n, quantum);
    return 0;
}
```
OUTPUT:


```
Enter the number of processes: 5
Enter burst time for process 1: 20
Enter burst time for process 2: 5
Enter burst time for process 3: 4
Enter burst time for process 4: 3
Enter burst time for process 5: 2
Enter time quantum: 2

Round Robin Scheduling:
Process 1 runs for 2 units.
Process 2 runs for 2 units.
Process 3 runs for 2 units.
Process 4 runs for 2 units.
Process 5 runs for 2 units.
Process 5 has completed.
Process 1 runs for 2 units.
Process 2 runs for 2 units.
Process 3 runs for 2 units.
Process 3 has completed.
Process 4 runs for 1 units.
Process 4 has completed.
Process 1 runs for 2 units.
Process 2 runs for 1 units.
Process 2 has completed.
Process 1 runs for 2 units.
Process 1 runs for 2 units.
Process 1 runs for 2 units.
Process 1 runs for 2 units.
Process 1 runs for 2 units.
Process 1 runs for 2 units.
Process 1 runs for 2 units.
Process 1 has completed.

--------------------------------
Process exited after 14.32 seconds with return value 0
Press any key to continue . . .
```

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

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);

}
```



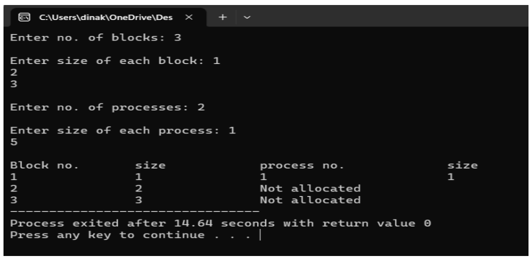
```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
int i;
void *shared_memory;
char buff[100];
int shmid;
shmid=shmget((key_t)2345, 1024, 0666|IPC_CREAT);
printf("Key of shared memory is %d\n",shmid);
shared_memory=shmat(shmid,NULL,0);
printf("Process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

```
/tmp/1gAS60mgvD.o
Key of shared memory is 0
Process attached at 0x7f99fe5b9000
Enter some data to write to shared memory
21
You wrote : 21

25
dash: 2: 25: not found
```

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

Program:

```
#include <stdio.h>
struct clientData
{
unsigned int acctNum;
char lastName[ 15 ];
char firstName[ 10 ];
double balance;
};
int main( void )
{
unsigned int i;
struct clientData blankClient = { 0, "", "", 0.0 };
FILE *cfPtr;
if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL )
{
puts( "File could not be opened." );
}
else
{
for ( i = 1; i <= 100; ++i )
{
fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
}
fclose ( cfPtr );
}
}
```

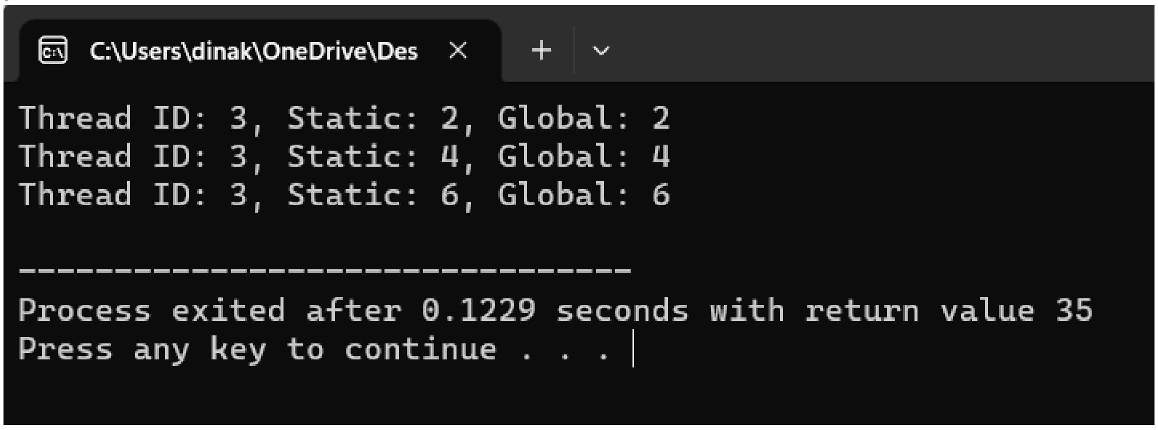


```
Enter no. of blocks: 3

Enter size of each block: 1
2
3

Enter no. of processes: 2

Enter size of each process: 1
5

Block no.       size            process no.             size
1               1               1                       1
2               2               Not allocated
3               3               Not allocated
--------------------------------
Process exited after 14.64 seconds with return value 0
Press any key to continue . . .
```

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

Program:

```
#include<stdio.h>
#include <pthread.h>
int g = 0;
void *myThreadFun(void *vargp)
{ int *myid = (int *)vargp;
static int s = 0; ++s; ++g;
printf("Thread ID: %d, Static: %d, Global: %d\n", *myid, ++s, ++g);
}
int main()
{           int i;
            pthread_t tid;
            for (i = 0; i < 3; i++)
            pthread_create(&tid, NULL, myThreadFun, (void *)&tid);
            pthread_exit(NULL);
}
```



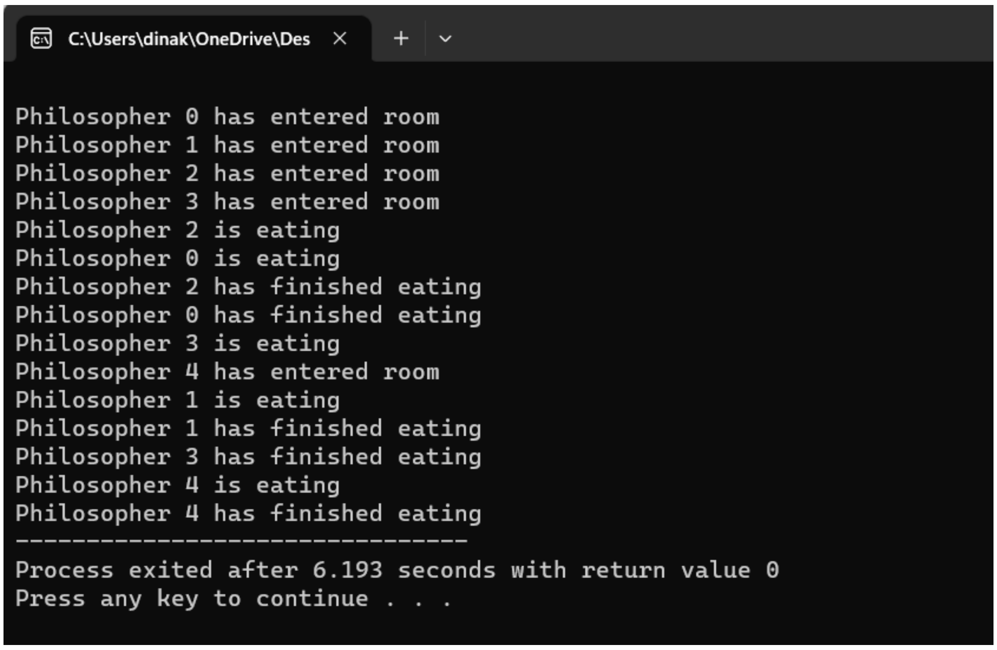
```
Thread ID: 3, Static: 2, Global: 2
Thread ID: 3, Static: 4, Global: 4
Thread ID: 3, Static: 6, Global: 6

--------------------------------
Process exited after 0.1229 seconds with return value 35
Press any key to continue . . .
```

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

Program:

```
#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);
}
```



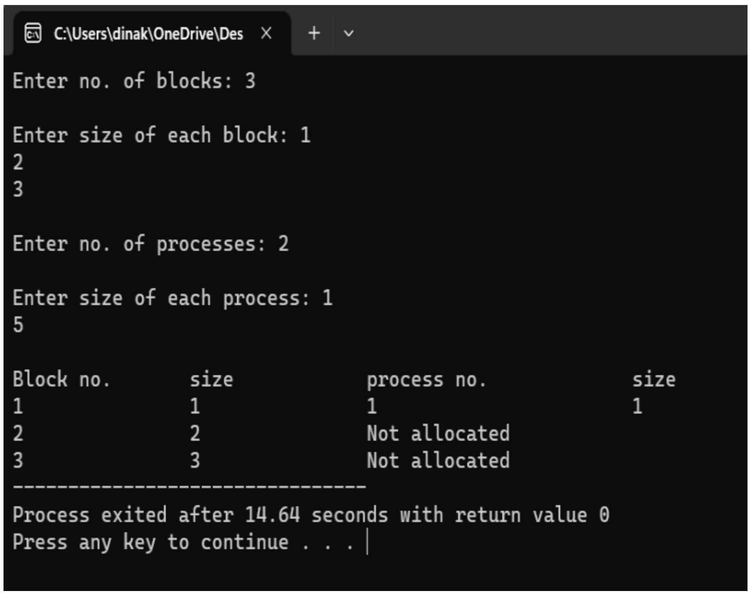
```
Philosopher 0 has entered room
Philosopher 1 has entered room
Philosopher 2 has entered room
Philosopher 3 has entered room
Philosopher 2 is eating
Philosopher 0 is eating
Philosopher 2 has finished eating
Philosopher 0 has finished eating
Philosopher 3 is eating
Philosopher 4 has entered room
Philosopher 1 is eating
Philosopher 1 has finished eating
Philosopher 3 has finished eating
Philosopher 4 is eating
Philosopher 4 has finished eating
--------------------------------
Process exited after 6.193 seconds with return value 0
Press any key to continue . . .
```

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

Program:

```
#include<stdio.h>

int 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");
        }
}
```

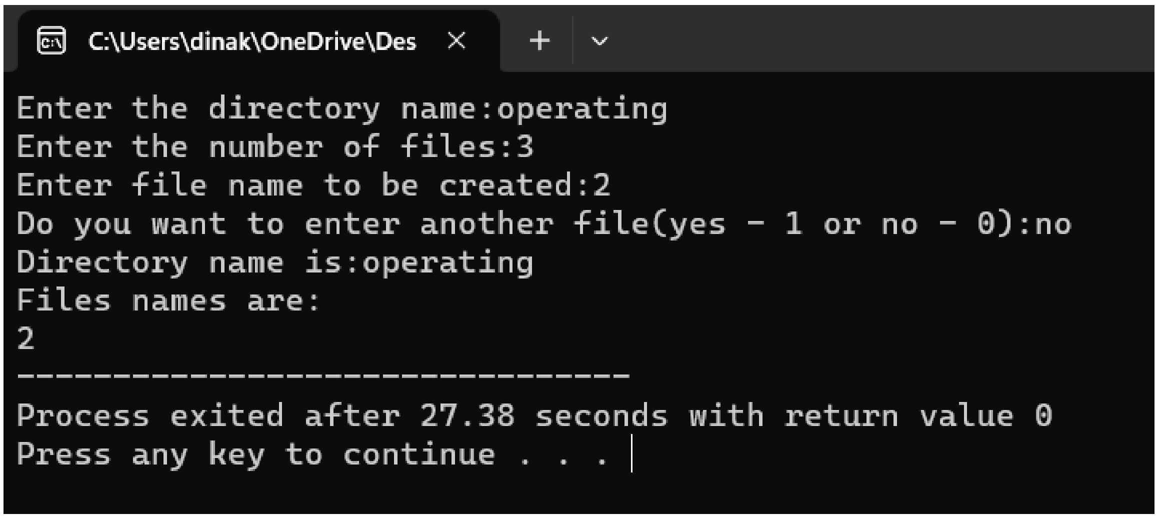


```
C:\Users\dinak\OneDrive\Des
Enter no. of blocks: 3

Enter size of each block: 1
2
3

Enter no. of processes: 2

Enter size of each process: 1
5

Block no.       size            process no.             size
1               1               1                       1
2               2               Not allocated
3               3               Not allocated
--------------------------------
Process exited after 14.64 seconds with return value 0
Press any key to continue . . .
```

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

Program:

```
#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();
}
```

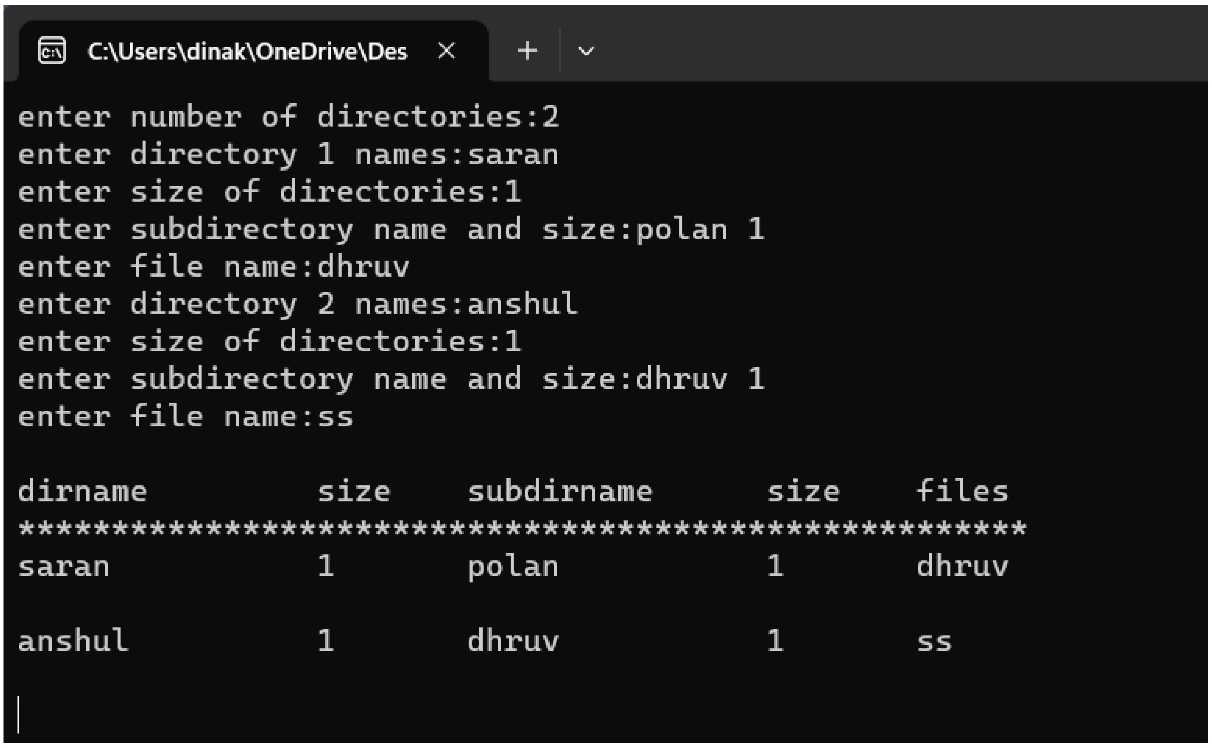


```
Enter the directory name:operating
Enter the number of files:3
Enter file name to be created:2
Do you want to enter another file(yes - 1 or no - 0):no
Directory name is:operating
Files names are:
2
--------------------------------
Process exited after 27.38 seconds with return value 0
Press any key to continue . . .
```

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

Program:

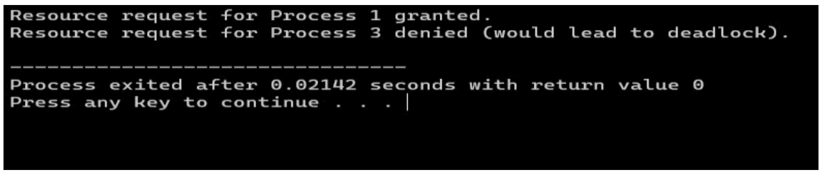
```
#include<stdio.h>

#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();
}
```



```
enter number of directories:2
enter directory 1 names:saran
enter size of directories:1
enter subdirectory name and size:polan 1
enter file name:dhruv
enter directory 2 names:anshul
enter size of directories:1
enter subdirectory name and size:dhruv 1
enter file name:ss

dirname         size    subdirname      size    files
******************************************************
saran           1       polan           1       dhruv

anshul          1       dhruv           1       ss
```

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

```
#include <stdio.h>
struct clientData
{
unsigned int acctNum;
char lastName[ 15 ];
char firstName[ 10 ];
double balance;
};
int main( void )
{
unsigned int i;
struct clientData blankClient = { 0, "", "", 0.0 };
FILE *cfPtr;
if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL )
{
puts( "File could not be opened." );
}
else
{
for ( i = 1; i <= 100; ++i )
{
fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
}
fclose ( cfPtr );
}
}
```



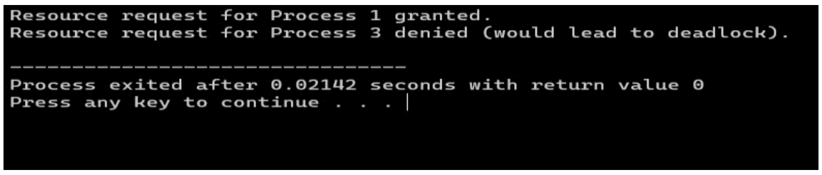
```
Resource request for Process 1 granted.
Resource request for Process 3 denied (would lead to deadlock).

--------------------------------
Process exited after 0.02142 seconds with return value 0
Press any key to continue . . .
```

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

Program:

```
#include <stdio.h>

// Define the number of processes and resources
#define NUM_PROCESSES 5
#define NUM_RESOURCES 3

// Available resources
int available[NUM_RESOURCES] = {3, 3, 2};

// Maximum demand of each process
int max_demand[NUM_PROCESSES][NUM_RESOURCES] = {
  {7, 5, 3},
  {3, 2, 2},
  {9, 0, 2},
  {2, 2, 2},
  {4, 3, 3}
};

// Allocated resources to processes
int allocation[NUM_PROCESSES][NUM_RESOURCES] = {
  {0, 1, 0},
  {2, 0, 0},
  {3, 0, 2},
  {2, 1, 1},
  {0, 0, 2}
};

// Need resources for each process
int need[NUM_PROCESSES][NUM_RESOURCES];

// Function to check if the system is in a safe state
int isSafeState(int process, int request[]) {
  // Step 1: Check if the request is less than or equal to the need
  for (int i = 0; i < NUM_RESOURCES; i++) {
    if (request[i] > need[process][i]) {
      return 0; // Request exceeds need
    }
  }

  // Step 2: Check if the request is less than or equal to the available resources
  for (int i = 0; i < NUM_RESOURCES; i++) {
    if (request[i] > available[i]) {
      return 0; // Request exceeds available resources
    }
  }
```

```
    // Try to allocate the resources and see if it remains safe
    for (int i = 0; i < NUM_RESOURCES; i++) {
        available[i] -= request[i];
        allocation[process][i] += request[i];
        need[process][i] -= request[i];
    }

    int finish[NUM_PROCESSES] = {0};
    int work[NUM_RESOURCES];
    for (int i = 0; i < NUM_RESOURCES; i++) {
        work[i] = available[i];
    }

    int count = 0;
    while (count < NUM_PROCESSES) {
        int found = 0;
        for (int i = 0; i < NUM_PROCESSES; i++) {
            if (finish[i] == 0) {
                int j;
                for (j = 0; j < NUM_RESOURCES; j++) {
                    if (need[i][j] > work[j]) {
                        break;
                    }
                }
                if (j == NUM_RESOURCES) {
                    for (j = 0; j < NUM_RESOURCES; j++) {
                        work[j] += allocation[i][j];
                    }
                    finish[i] = 1;
                    found = 1;
                    count++;
                }
            }
        }
        if (!found) {
            for (int i = 0; i < NUM_RESOURCES; i++) {
                available[i] += request[i];
                allocation[process][i] -= request[i];
                need[process][i] += request[i];
            }
            return 0; // System is not in a safe state
        }
    }

    // If we reach here, the system is in a safe state
    return 1;
}

// Function to allocate resources to a process
void allocateResources(int process, int request[]) {
    if (isSafeState(process, request)) {
        printf("Resource request for Process %d granted.\n", process);
    } else {
        printf("Resource request for Process %d denied (would lead to deadlock).\n", process);
    }
}

int main() {
    // Initialize the 'need' matrix
    for (int i = 0; i < NUM_PROCESSES; i++) {
        for (int j = 0; j < NUM_RESOURCES; j++) {
            need[i][j] = max_demand[i][j] - allocation[i][j];
        }
    }

    // Test resource allocation requests
    int process = 1;
    int request[NUM_RESOURCES] = {1, 0, 2};
    allocateResources(process, request);

    process = 3;
    int request2[NUM_RESOURCES] = {0, 2, 0};
    allocateResources(process, request2);

    return 0;
}
```



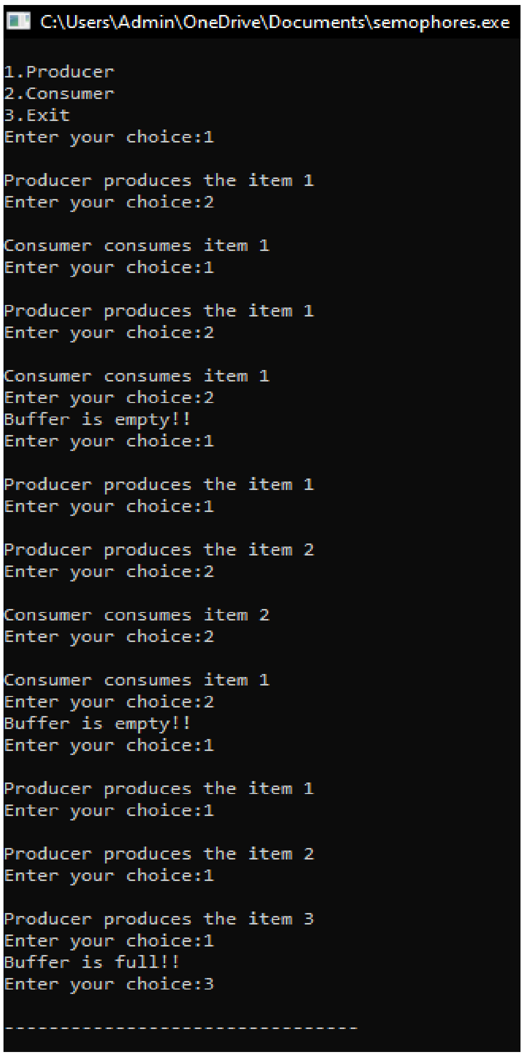
```
Resource request for Process 1 granted.
Resource request for Process 3 denied (would lead to deadlock).

--------------------------------
Process exited after 0.02142 seconds with return value 0
Press any key to continue . . .
```

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

Program:

```
#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);
}
```

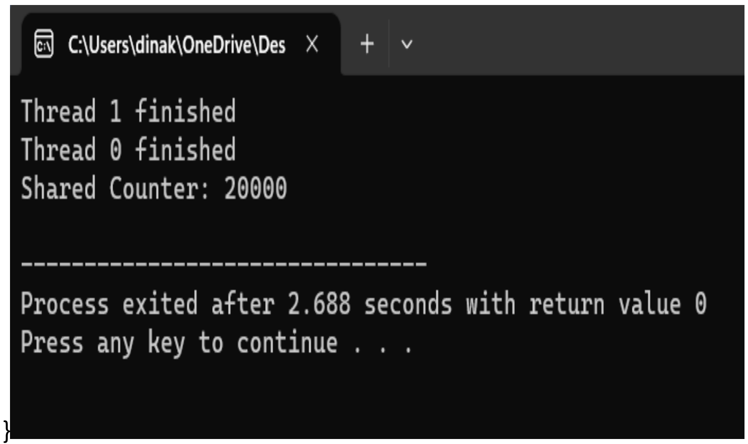


```
1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces the item 1
Enter your choice:2

Consumer consumes item 1
Enter your choice:1

Producer produces the item 1
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:1

Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:3

--------------------------------
```

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

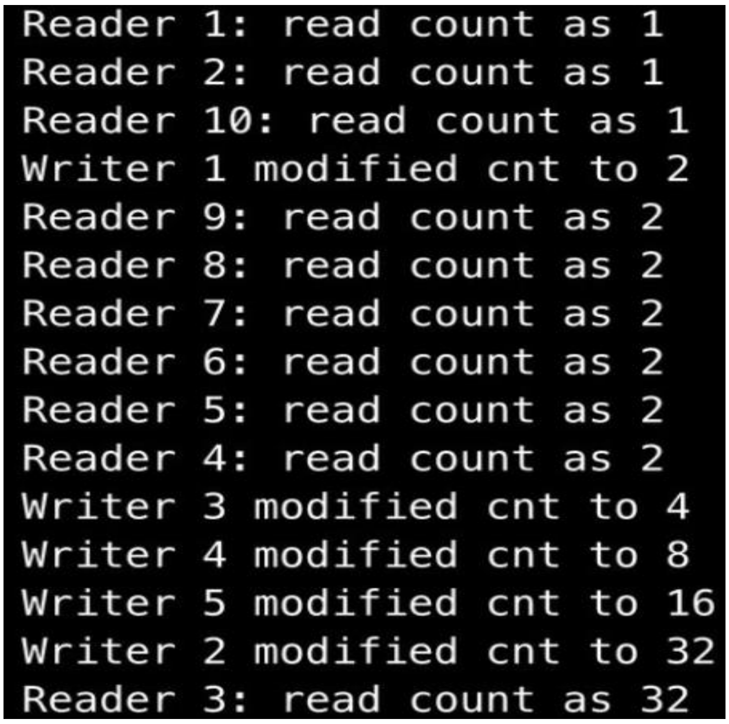
Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define NUM_THREADS 2
#define NUM_INCREMENTS 10000
int sharedCounter = 0;
pthread_mutex_t mutex;
void* incrementCounter(void* threadID) {
   int id = *((int*)threadID);
for (int i = 0; i < NUM_INCREMENTS; i++) {
      pthread_mutex_lock(&mutex);
      sharedCounter++;
      pthread_mutex_unlock(&mutex);
   }
printf("Thread %d finished\n", id);
   pthread_exit(NULL);
}
int main() {
   pthread_t threads[NUM_THREADS];
   int threadIDs[NUM_THREADS];
```

```
pthread_mutex_init(&mutex, NULL);
for (int i = 0; i < NUM_THREADS; i++) {
      threadIDs[i] = i;
      int result = pthread_create(&threads[i], NULL, incrementCounter, &threadIDs[i]);
      if (result) {
         fprintf(stderr, "Error creating thread %d: %d\n", i, result);
         exit(-1);
      }
   }
for (int i = 0; i < NUM_THREADS; i++) {
      pthread_join(threads[i], NULL);}
pthread_mutex_destroy(&mutex);
printf("Shared Counter: %d\n", sharedCounter);
   return 0;
```

```
C:\Users\dinak\OneDrive\Des

Thread 1 finished
Thread 0 finished
Shared Counter: 20000

--------------------------------
Process exited after 2.688 seconds with return value 0
Press any key to continue . . .
```
```
}
```
20.Construct a C program to simulate Reader-Writer problem using Semaphores.

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

#define NUM_READERS 5
#define NUM_WRITERS 2

int shared_data = 0;
sem_t mutex, writeblock;
int reader_count = 0;

void *reader(void *arg) {
   while (1) {
      usleep(100000);  // Sleep for a short time to simulate reading
      sem_wait(&mutex);
      reader_count++;
      if (reader_count == 1) {
         sem_wait(&writeblock);
      }
      sem_post(&mutex);

      // Reading the shared data
      printf("Reader is reading: %d\n", shared_data);

      sem_wait(&mutex);
```

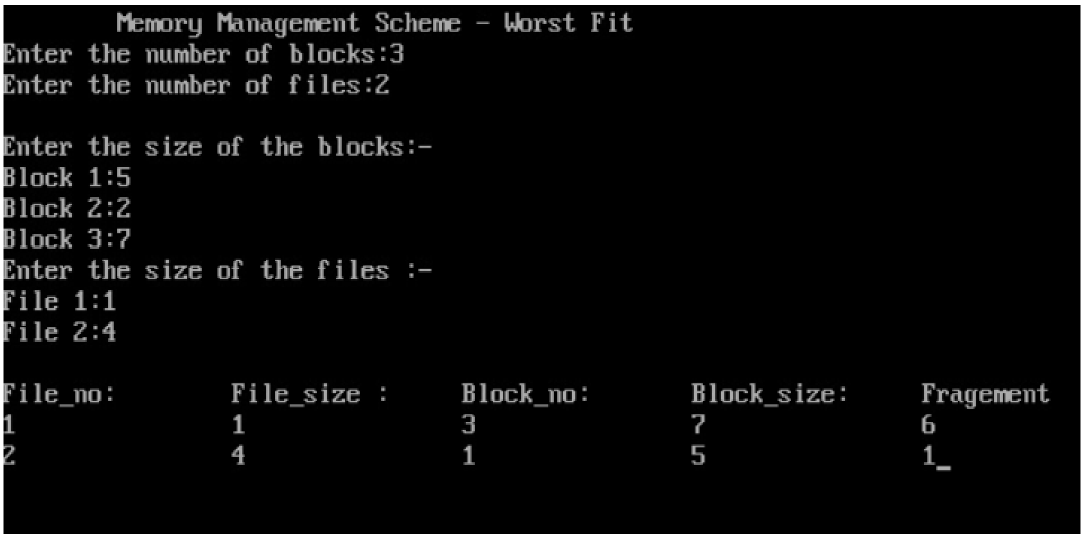
```
        reader_count--;
        if (reader_count == 0) {
            sem_post(&writeblock);
        }
        sem_post(&mutex);
    }
}

void *writer(void *arg) {
    while (1) {
        sem_wait(&writeblock);
        // Modifying the shared data
        shared_data++;
        printf("Writer is writing: %d\n", shared_data);
        sem_post(&writeblock);

        usleep(100000);  // Sleep for a short time to simulate writing
    }
}

int main() {
    pthread_t reader_threads[NUM_READERS];
    pthread_t writer_threads[NUM_WRITERS];

    sem_init(&mutex, 0, 1);
    sem_init(&writeblock, 0, 1);

    for (int i = 0; i < NUM_READERS; i++) {
        pthread_create(&reader_threads[i], NULL, reader, NULL);
    }

    for (int i = 0; i < NUM_WRITERS; i++) {
        pthread_create(&writer_threads[i], NULL, writer, NULL);
    }

    for (int i = 0; i < NUM_READERS; i++) {
        pthread_join(reader_threads[i], NULL);
    }

    for (int i = 0; i < NUM_WRITERS; i++) {
        pthread_join(writer_threads[i], NULL);
    }

    sem_destroy(&mutex);
    sem_destroy(&writeblock);

    return 0;
}
```



```
Reader 1: read count as 1
Reader 2: read count as 1
Reader 10: read count as 1
Writer 1 modified cnt to 2
Reader 9: read count as 2
Reader 8: read count as 2
Reader 7: read count as 2
Reader 6: read count as 2
Reader 5: read count as 2
Reader 4: read count as 2
Writer 3 modified cnt to 4
Writer 4 modified cnt to 8
Writer 5 modified cnt to 16
Writer 2 modified cnt to 32
Reader 3: read count as 32
```

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

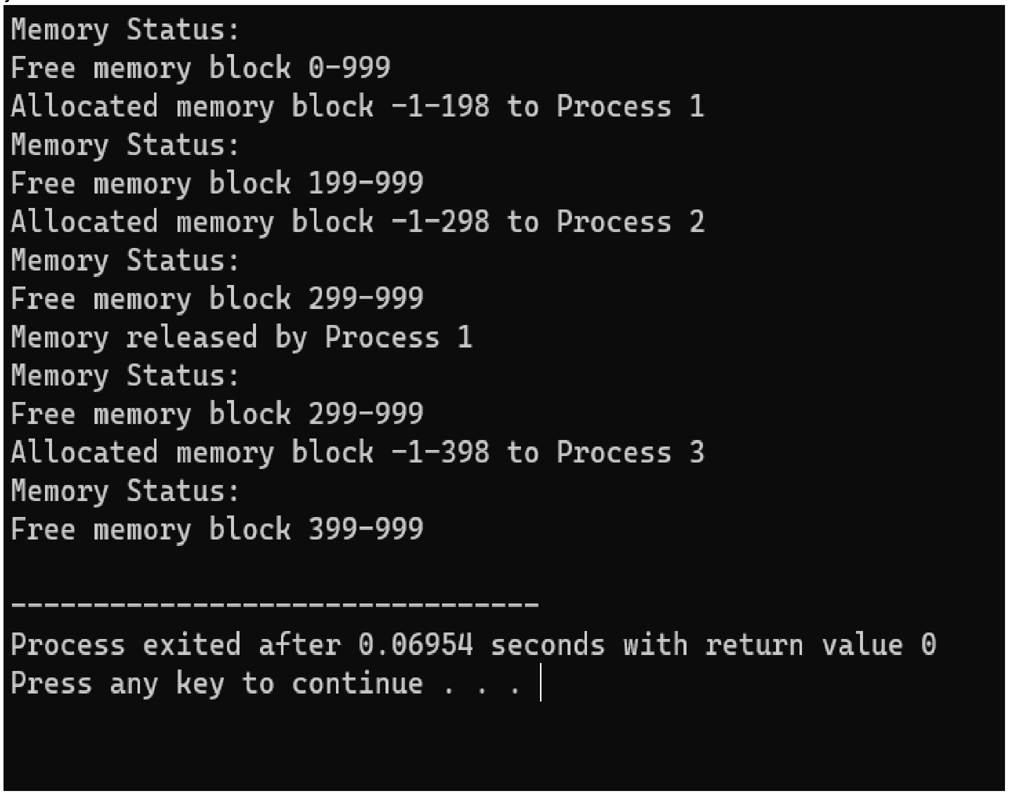
```c
#include <stdio.h>

#define MAX_MEMORY 1000

int memory[MAX_MEMORY];

// Function to initialize memory
void initializeMemory() {
  for (int i = 0; i < MAX_MEMORY; i++) {
    memory[i] = -1; // -1 indicates that the memory is unallocated
  }
}

// Function to display memory status
void displayMemory() {
  int i, j;
  int count = 0;
  printf("Memory Status:\n");

  for (i = 0; i < MAX_MEMORY; i++) {
    if (memory[i] == -1) {
      count++;
      j = i;
      while (memory[j] == -1 && j < MAX_MEMORY) {
        j++;
      }
      printf("Free memory block %d-%d\n", i, j - 1);
      i = j - 1;
    }
  }

  if (count == 0) {
        printf("No free memory available.\n");
    }
}

// Function to allocate memory using worst-fit algorithm
void allocateMemory(int processId, int size) {
    int start = -1;
    int blockSize = 0;

    for (int i = 0; i < MAX_MEMORY; i++) {
        if (memory[i] == -1) {
            if (blockSize == 0) {
                start = i;
            }
            blockSize++;
        } else {
            blockSize = 0;
        }

        if (blockSize >= size) {
            break;
        }
    }

    if (blockSize >= size) {
        for (int i = start; i < start + size; i++) {
            memory[i] = processId;
        }
        printf("Allocated memory block %d-%d to Process %d\n", start, start + size - 1, processId);
    } else {
        printf("Memory allocation for Process %d failed (not enough contiguous memory).\n", processId);
    }
}

// Function to deallocate memory
void deallocateMemory(int processId) {
    for (int i = 0; i < MAX_MEMORY; i++) {
        if (memory[i] == processId) {
            memory[i] = -1;
        }
    }
    printf("Memory released by Process %d\n", processId);
}

int main() {
    initializeMemory();
    displayMemory();

    allocateMemory(1, 200);
    displayMemory();

    allocateMemory(2, 300);
    displayMemory();

    deallocateMemory(1);
    displayMemory();
    allocateMemory(3, 400);
    displayMemory();

    return 0;
}
```



```
       Memory Management Scheme - Worst Fit
Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

File_no:        File_size :     Block_no:       Block_size:     Fragement
1               1               3               7               6
2               4               1               5               1_
```

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

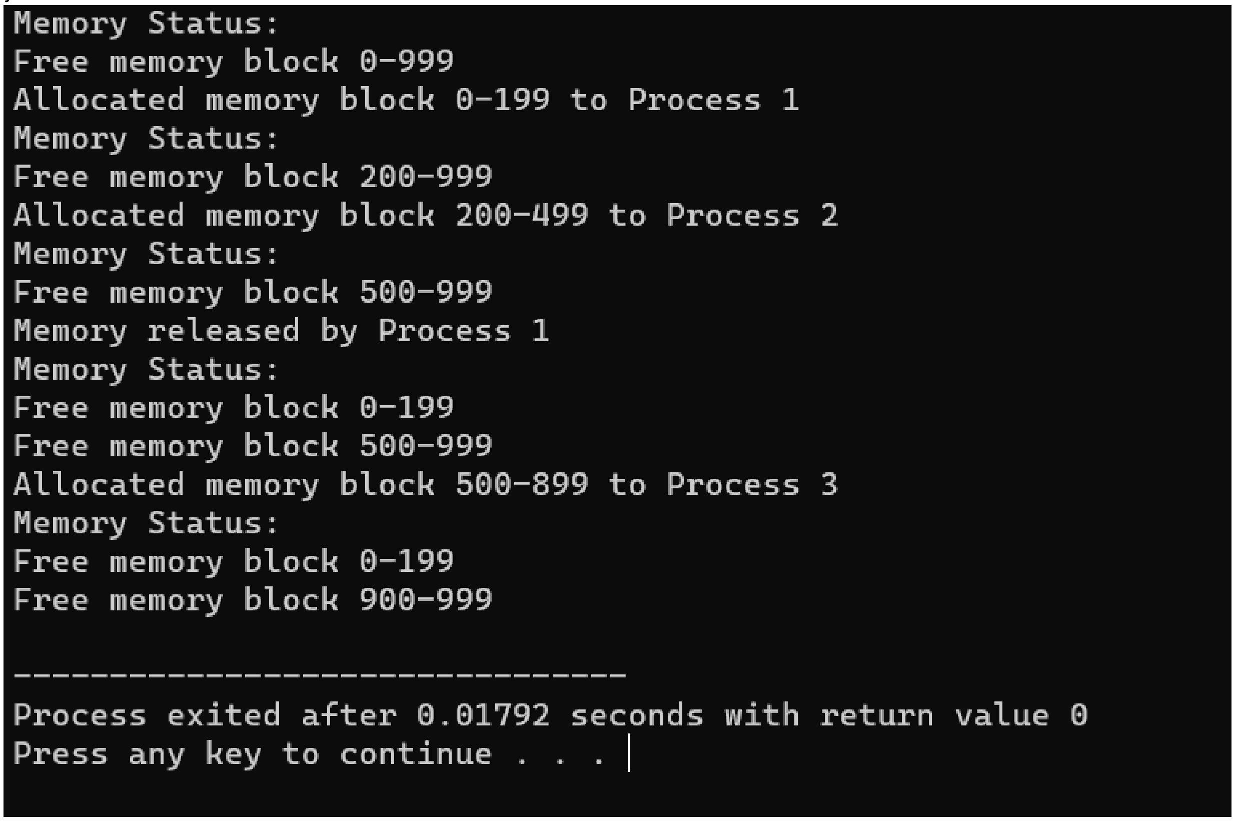
```
#include <stdio.h>

#define MAX_MEMORY 1000

int memory[MAX_MEMORY];

// Function to initialize memory
void initializeMemory() {
    for (int i = 0; i < MAX_MEMORY; i++) {
        memory[i] = -1; // -1 indicates that the memory is unallocated
    }
}

// Function to display memory status
void displayMemory() {
    int i, j;
    int count = 0;
    printf("Memory Status:\n");

    for (i = 0; i < MAX_MEMORY; i++) {
        if (memory[i] == -1) {
            count++;
            j = i;
            while (memory[j] == -1 && j < MAX_MEMORY) {
                j++;
            }
            printf("Free memory block %d-%d\n", i, j - 1);
            i = j - 1;
        }
    }
    if (count == 0) {
        printf("No free memory available.\n");
    }
}

// Function to allocate memory using best-fit algorithm
void allocateMemory(int processId, int size) {
    int start = -1;
    int blockSize = MAX_MEMORY;
    int bestStart = -1;
    int bestSize = MAX_MEMORY;

    for (int i = 0; i < MAX_MEMORY; i++) {
        if (memory[i] == -1) {
            if (blockSize == MAX_MEMORY) {
                start = i;
            }
            blockSize++;
        } else {
            if (blockSize >= size && blockSize < bestSize) {
                bestSize = blockSize;
                bestStart = start;
            }
            blockSize = 0;
        }
    }

    if (bestSize >= size) {
        for (int i = bestStart; i < bestStart + size; i++) {
            memory[i] = processId;
        }
        printf("Allocated memory block %d-%d to Process %d\n", bestStart, bestStart + size - 1, processId);
    } else {
        printf("Memory allocation for Process %d failed (not enough contiguous memory).\n", processId);
    }
}

// Function to deallocate memory
void deallocateMemory(int processId) {
    for (int i = 0; i < MAX_MEMORY; i++) {
        if (memory[i] == processId) {
            memory[i] = -1;
        }
    }
    printf("Memory released by Process %d\n", processId);
}

int main() {
    initializeMemory();
    displayMemory();

    allocateMemory(1, 200);
    displayMemory();

    allocateMemory(2, 300);
    displayMemory();
    deallocateMemory(1);
    displayMemory();

    allocateMemory(3, 400);
    displayMemory();

    return 0;
}
```



```
Memory Status:
Free memory block 0-999
Allocated memory block -1-198 to Process 1
Memory Status:
Free memory block 199-999
Allocated memory block -1-298 to Process 2
Memory Status:
Free memory block 299-999
Memory released by Process 1
Memory Status:
Free memory block 299-999
Allocated memory block -1-398 to Process 3
Memory Status:
Free memory block 399-999

--------------------------------
Process exited after 0.06954 seconds with return value 0
Press any key to continue . . .
```

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

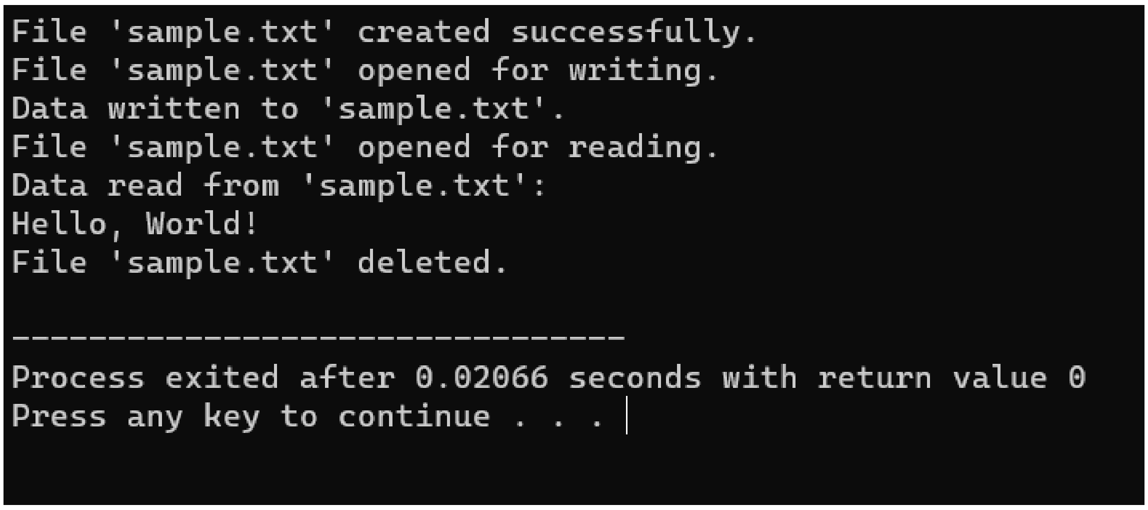
```
#include <stdio.h>

#define MAX_MEMORY 1000

int memory[MAX_MEMORY];

// Function to initialize memory
void initializeMemory() {
    for (int i = 0; i < MAX_MEMORY; i++) {
        memory[i] = -1; // -1 indicates that the memory is unallocated
    }
}

// Function to display memory status
void displayMemory() {
    int i, j;
    int count = 0;
    printf("Memory Status:\n");

    for (i = 0; i < MAX_MEMORY; i++) {
        if (memory[i] == -1) {
            count++;
            j = i;
            while (memory[j] == -1 && j < MAX_MEMORY) {
                j++;
            }
            printf("Free memory block %d-%d\n", i, j - 1);
            i = j - 1;
        }
    }

    if (count == 0) {
        printf("No free memory available.\n");
    }
}

// Function to allocate memory using first-fit algorithm
void allocateMemory(int processId, int size) {
    int start = -1;
    int blockSize = 0;

    for (int i = 0; i < MAX_MEMORY; i++) {
        if (memory[i] == -1) {
            if (blockSize == 0) {
                start = i;
            }
            blockSize++;
        } else {
            blockSize = 0;
        }

        if (blockSize >= size) {
            break;
        }
    }

    if (blockSize >= size) {
        for (int i = start; i < start + size; i++) {
            memory[i] = processId;
        }
        printf("Allocated memory block %d-%d to Process %d\n", start, start + size - 1, processId);
    } else {
        printf("Memory allocation for Process %d failed (not enough contiguous memory).\n", processId);
    }
}

// Function to deallocate memory
void deallocateMemory(int processId) {
    for (int i = 0; i < MAX_MEMORY; i++) {
        if (memory[i] == processId) {
            memory[i] = -1;
        }
    }
    printf("Memory released by Process %d\n", processId);
}

int main() {
    initializeMemory();
    displayMemory();
    allocateMemory(1, 200);
    displayMemory();

    allocateMemory(2, 300);
    displayMemory();

    deallocateMemory(1);
    displayMemory();

    allocateMemory(3, 400);
    displayMemory();

    return 0;
}
```



```
Memory Status:
Free memory block 0-999
Allocated memory block 0-199 to Process 1
Memory Status:
Free memory block 200-999
Allocated memory block 200-499 to Process 2
Memory Status:
Free memory block 500-999
Memory released by Process 1
Memory Status:
Free memory block 0-199
Free memory block 500-999
Allocated memory block 500-899 to Process 3
Memory Status:
Free memory block 0-199
Free memory block 900-999

--------------------------------
Process exited after 0.01792 seconds with return value 0
Press any key to continue . . .
```

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

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

int main() {
    int fd;
    char buffer[100];

    // Creating a new file
    fd = creat("sample.txt", S_IRWXU);
    if (fd == -1) {
        perror("create");
        exit(1);
    } else {
        printf("File 'sample.txt' created successfully.\n");
        close(fd);
    }

    // Opening an existing file for writing
    fd = open("sample.txt", O_WRONLY | O_APPEND);
    if (fd == -1) {
        perror("open");
        exit(1);
    } else {
        printf("File 'sample.txt' opened for writing.\n");
    }

    // Writing data to the file
    write(fd, "Hello, World!\n", 14);
    printf("Data written to 'sample.txt'.\n");
    close(fd);

    // Opening the file for reading
    fd = open("sample.txt", O_RDONLY);
    if (fd == -1) {
        perror("open");
        exit(1);
    } else {
        printf("File 'sample.txt' opened for reading.\n");
    }

    // Reading data from the file
    int bytesRead = read(fd, buffer, sizeof(buffer));
    if (bytesRead == -1) {
        perror("read");
        exit(1);
    } else {
        printf("Data read from 'sample.txt':\n");
        write(STDOUT_FILENO, buffer, bytesRead);
    }
    close(fd);

    // Deleting the file
    if (remove("sample.txt") == -1) {
        perror("remove");
        exit(1);
    } else {
        printf("File 'sample.txt' deleted.\n");
    }

    return 0;
}
```

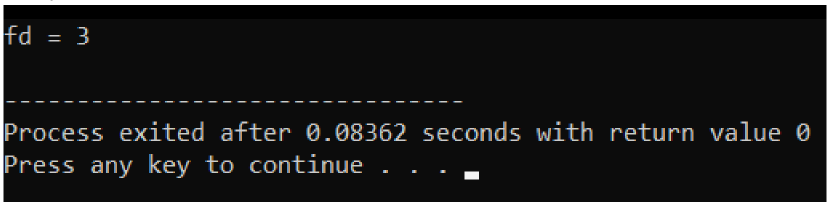

```
File 'sample.txt' created successfully.
File 'sample.txt' opened for writing.
Data written to 'sample.txt'.
File 'sample.txt' opened for reading.
Data read from 'sample.txt':
Hello, World!
File 'sample.txt' deleted.

--------------------------------
Process exited after 0.02066 seconds with return value 0
Press any key to continue . . .
```

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

Program:

```
#include<stdio.h>
#include<fcntl.h>
#include<errno.h>
extern int errno;
int main()
{

        int fd = open("foo.txt", O_RDONLY | O_CREAT);
        printf("fd = %d\n", fd);
        if (fd ==-1)
        {
                printf("Error Number % d\n", errno);
                perror("Program");
        }
        return 0;
}
```

Output:


```
fd = 3

--------------------------------
Process exited after 0.08362 seconds with return value 0
Press any key to continue . . .
```

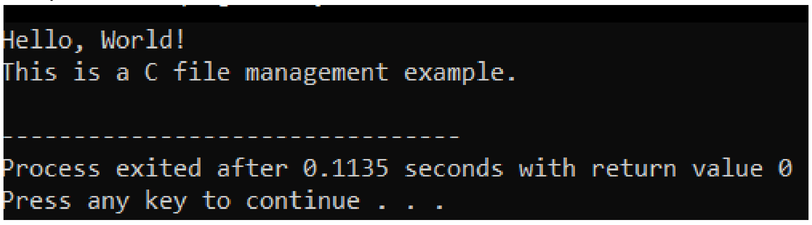
26) Construct a C program to implement the file management operations.

Program:

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

```
int main() {
   FILE *file;
   file = fopen("example.txt", "w");
   if (file == NULL) {
      printf("Error opening the file for writing.\n");
      return 1;
   }
   fprintf(file, "Hello, World!\n");
   fprintf(file, "This is a C file management example.\n");
   fclose(file);
   file = fopen("example.txt", "r");
   if (file == NULL) {
      printf("Error opening the file for reading.\n");
      return 1;
   }
   char buffer[100];
   while (fgets(buffer, sizeof(buffer), file) != NULL) {
      printf("%s", buffer);
   }
   fclose(file);

   return 0;
}
```

Output:



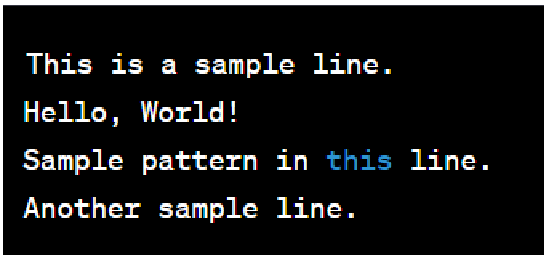
```
Hello, World!
This is a C file management example.

--------------------------------
Process exited after 0.1135 seconds with return value 0
Press any key to continue . . .
```

27) Develop a C program for simulating the function of ls UNIX Command.

Program:

```
#include<stdio.h>
#include<dirent.h>
   int main()
          {
      char fn[10], pat[10], temp[200];
      FILE *fp;
      printf("\n Enter file name : ");
      scanf("%s", fn);
      printf("Enter the pattern: ");
      scanf("%s", pat);
      fp = fopen(fn, "r");
      while (!feof(fp)) {
         fgets(temp, sizeof(fp), fp);
```

```
        if (strcmp(temp, pat))
            printf("%s", temp);
    }
    fclose(fp);
    return 1;

  }
```
Output:



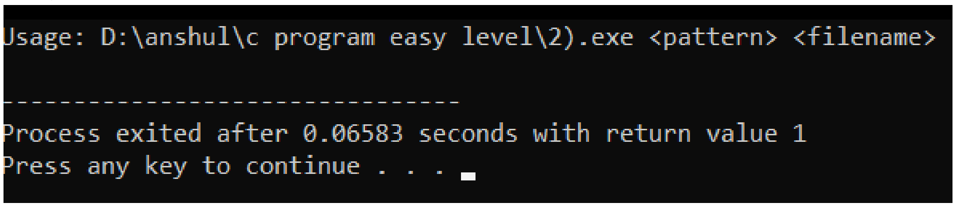
```
This is a sample line.
Hello, World!
Sample pattern in this line.
Another sample line.
```

28) Write a C program for simulation of GREP UNIX command

Program:

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

#define MAX_LINE_LENGTH 1024

void searchFile(const char *pattern, const char *filename) {
   FILE *file = fopen(filename, "r");
   if (file == NULL) {
      perror("Error opening file");
      exit(1);
   }

   char line[MAX_LINE_LENGTH];
   while (fgets(line, sizeof(line), file)) {
      if (strstr(line, pattern) != NULL) {
         printf("%s", line);
      }
   }

   fclose(file);
}

int main(int argc, char *argv[]) {
   if (argc != 3) {
      fprintf(stderr, "Usage: %s <pattern> <filename>\n", argv[0]);
      return 1;
   }

   const char *pattern = argv[1];
   const char *filename = argv[2];
```

```
    searchFile(pattern, filename);

    return 0;
}
```
Output:



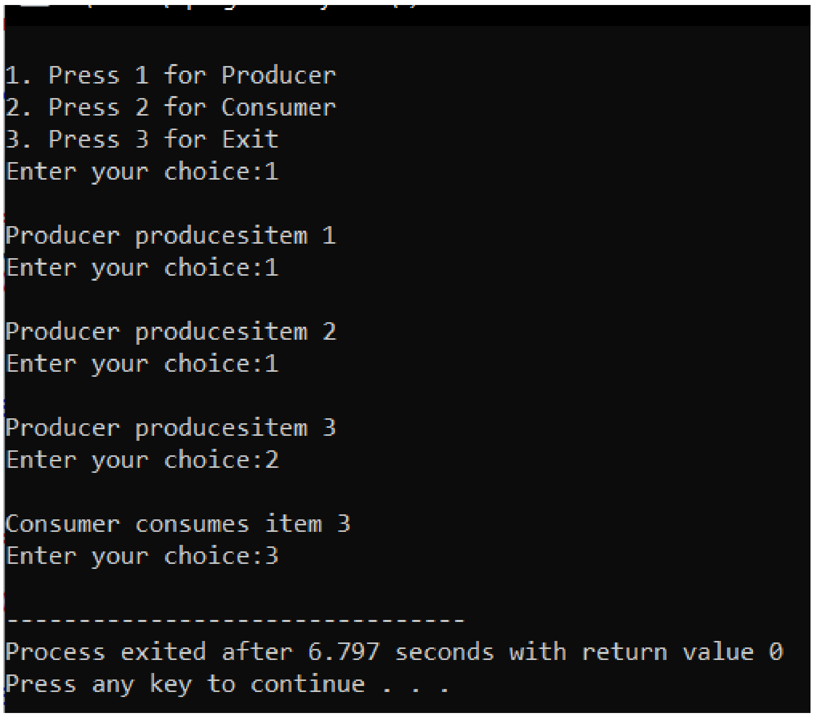
```
Usage: D:\anshul\c program easy level\2).exe <pattern> <filename>

--------------------------------
Process exited after 0.06583 seconds with return value 1
Press any key to continue . . .
```

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

Program:

```
#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;
      }
   }
}
```
Output:

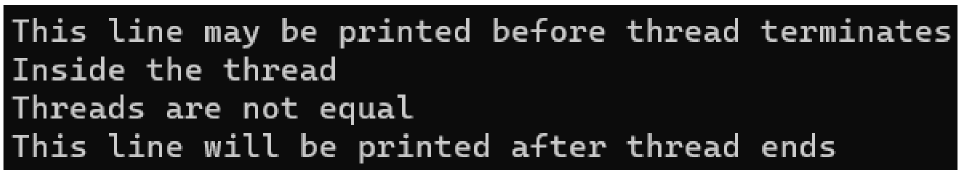


```
1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:1

Producer producesitem 1
Enter your choice:1

Producer producesitem 2
Enter your choice:1

Producer producesitem 3
Enter your choice:2

Consumer consumes item 3
Enter your choice:3

--------------------------------
Process exited after 6.797 seconds with return value 0
Press any key to continue . . .
```

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

```
#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()
{
```

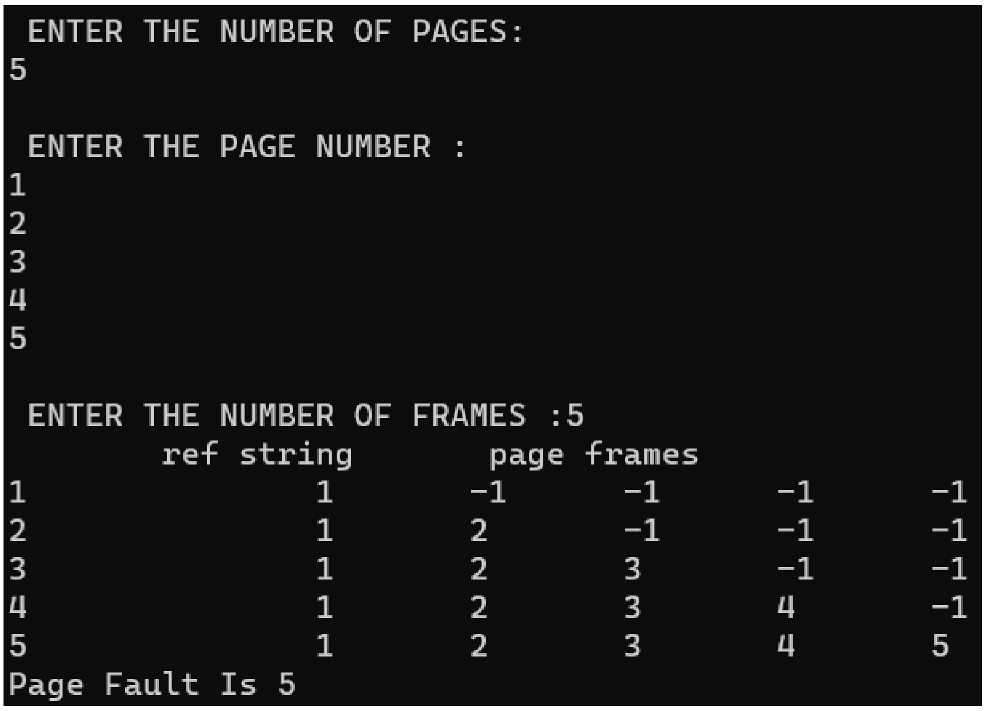
```
            fun();
            return 0;
}
```



```
This line may be printed before thread terminates
Inside the thread
Threads are not equal
This line will be printed after thread ends
```

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

```
#include<stdio.h>
int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
printf("\n ENTER THE PAGE NUMBER :\n");
for(i=1;i<=n;i++)
scanf("%d",&a[i]);
 printf("\n ENTER THE NUMBER OF FRAMES :");
scanf("%d",&no);
for(i=0;i<no;i++)
 frame[i]= -1;
 j=0;
 printf("\tref string\t page frames\n");
for(i=1;i<=n;i++)
                    {
 printf("%d\t\t",a[i]);
 avail=0;
for(k=0;k<no;k++)
if(frame[k]==a[i])
    avail=1;
    if (avail==0)
 {
  frame[j]=a[i];
  j=(j+1)%no;
  count++;
for(k=0;k<no;k++)
printf("%d\t",frame[k]);
}
  printf("\n");
}
printf("Page Fault Is %d",count);
 return 0;
}
```

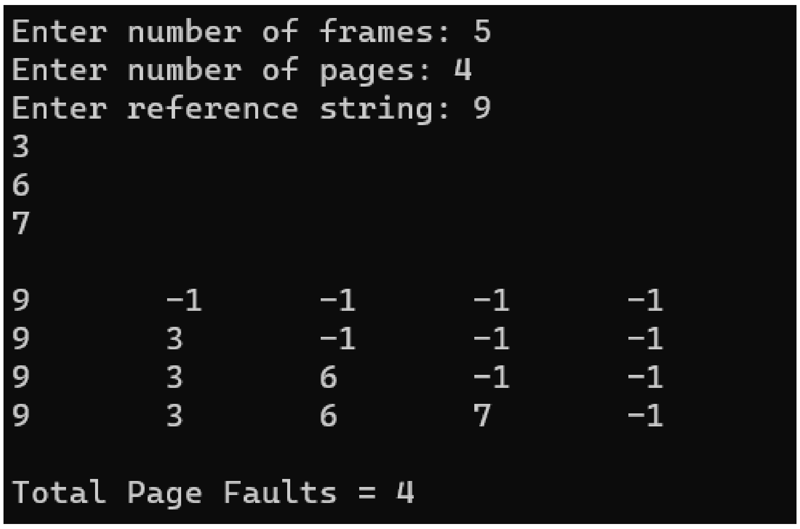


```
 ENTER THE NUMBER OF PAGES:
5

 ENTER THE PAGE NUMBER :
1
2
3
4
5

 ENTER THE NUMBER OF FRAMES :5
        ref string       page frames
1               1       -1      -1      -1      -1
2               1       2       -1      -1      -1
3               1       2       3       -1      -1
4               1       2       3       4       -1
5               1       2       3       4       5
Page Fault Is 5
```

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

```
#include<stdio.h>
int findLRU(int time[], int n)
{
int i, minimum = time[0], pos = 0;
for(i = 1; i< n; ++i)
{
if(time[i] < minimum)
{
minimum = time[i];
pos= i;
}
}
return pos;
}
int main()
{
int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos,
faults = 0;
printf("Enter number of frames: ");
scanf("%d", &no_of_frames);
printf("Enter number of pages: ");
scanf("%d", &no_of_pages);
printf("Enter reference string: ");
   for(i = 0; i<no_of_pages; ++i)
   {
scanf("%d", &pages[i]);
```

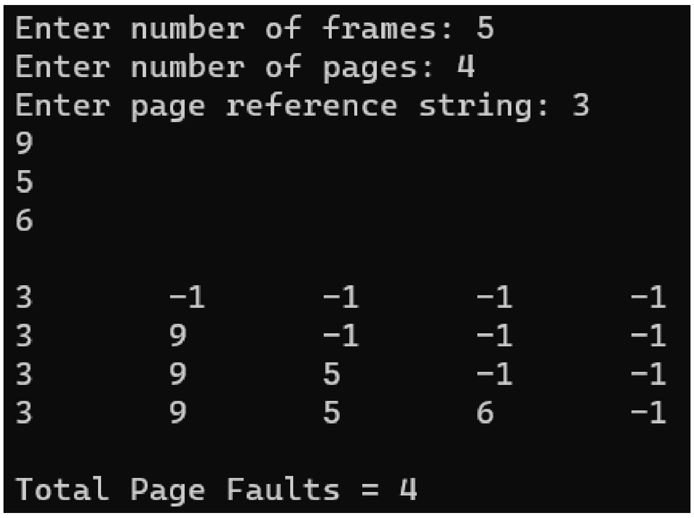
```
    }
for(i = 0; i<no_of_frames; ++i)
{
     frames[i] = -1;
    }
    for(i = 0; i<no_of_pages; ++i)
    {
     flag1 = flag2 = 0;
     for(j = 0; j <no_of_frames; ++j)
     {
     if(frames[j] == pages[i])
     {
     counter++;
     time[j] = counter;
   flag1 = flag2 = 1;
   break;
   }
     }
     if(flag1 == 0)
     {
for(j = 0; j <no_of_frames; ++j)
{
     if(frames[j] == -1)
     {
     counter++;
     faults++;
     frames[j] = pages[i];
     time[j] = counter;
     flag2 = 1;
     break;
     }
     }
     }
     if(flag2 == 0){
pos = findLRU(time, no_of_frames);
     counter++;
     faults++;
     frames[pos] = pages[i];
     time[pos] = counter;
     }
printf("\n");
     for(j = 0; j <no_of_frames; ++j){
printf("%d\t", frames[j]);
     }
}
printf("\n\nTotal Page Faults = %d", faults);
    return 0;
}
```


```
Enter number of frames: 5
Enter number of pages: 4
Enter reference string: 9
3
6
7

9       -1      -1      -1      -1
9       3       -1      -1      -1
9       3       6       -1      -1
9       3       6       7       -1

Total Page Faults = 4
```

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

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max,
faults = 0;
printf("Enter number of frames: ");
scanf("%d", &no_of_frames);
printf("Enter number of pages: ");
scanf("%d", &no_of_pages);
printf("Enter page reference string: ");
  for(i = 0; i<no_of_pages; ++i){
scanf("%d", &pages[i]);
  }
  for(i = 0; i<no_of_frames; ++i){
    frames[i] = -1;
  }
  for(i = 0; i<no_of_pages; ++i){
    flag1 = flag2 = 0;

    for(j = 0; j <no_of_frames; ++j){
      if(frames[j] == pages[i]){
          flag1 = flag2 = 1;
          break;
        }
    }
    if(flag1 == 0){
      for(j = 0; j <no_of_frames; ++j){
        if(frames[j] == -1){
          faults++;
          frames[j] = pages[i];
          flag2 = 1;
```

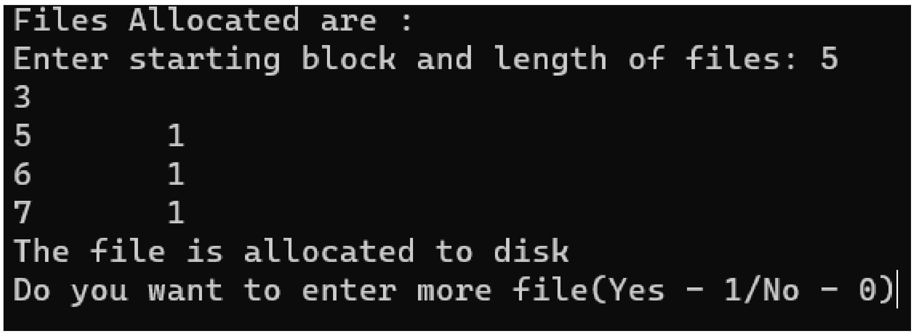
```
                    break;
                }
            }
        }
        if(flag2 == 0){
         flag3 =0;
            for(j = 0; j <no_of_frames; ++j){
             temp[j] = -1;

             for(k = i + 1; k <no_of_pages; ++k){
             if(frames[j] == pages[k]){
             temp[j] = k;
             break;
             }
             }
            }
            for(j = 0; j <no_of_frames; ++j){
             if(temp[j] == -1){
pos = j;
             flag3 = 1;
             break;
             }
            }
            if(flag3 ==0){
             max = temp[0];
pos = 0;

             for(j = 1; j <no_of_frames; ++j){
             if(temp[j] > max){
             max = temp[j];
pos = j;
             }
             }
            }
frames[pos] = pages[i];
faults++;
        }
printf("\n");
        for(j = 0; j <no_of_frames; ++j){
printf("%d\t", frames[j]);
        }
    }
printf("\n\nTotal Page Faults = %d", faults);
 return 0;
}
```



```
Enter number of frames: 5
Enter number of pages: 4
Enter page reference string: 3
9
5
6

3       -1      -1      -1      -1
3       9       -1      -1      -1
3       9       5       -1      -1
3       9       5       6       -1

Total Page Faults = 4
```

34. 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.

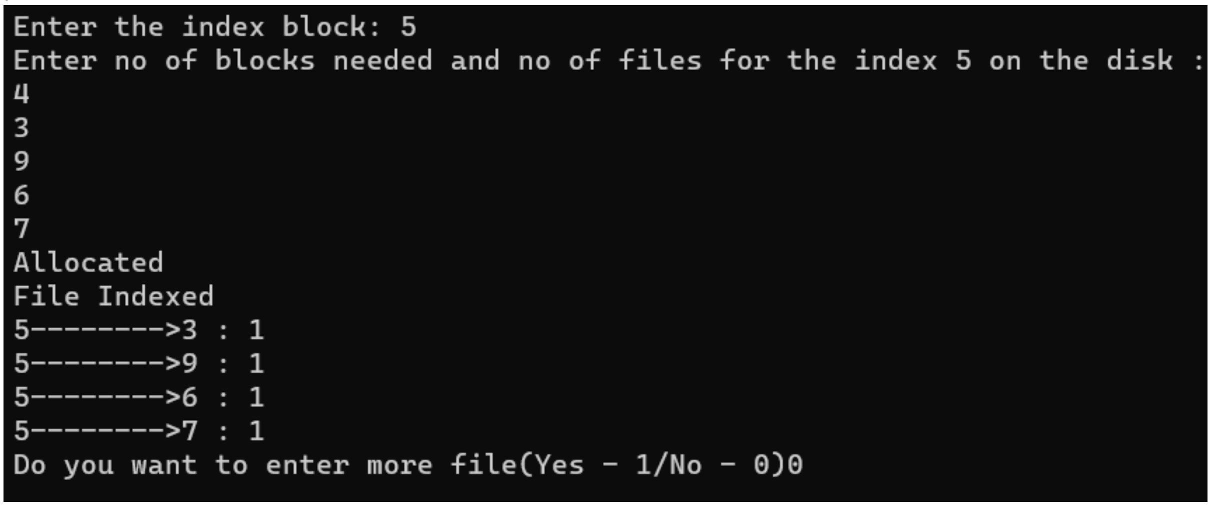
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
int f[50], i, st, len, j, c, k, count = 0;
for(i=0;i<50;i++)
f[i]=0;
printf("Files Allocated are : \n");
x : count=0;
printf("Enter starting block and length of files: ");
scanf("%d%d", &st,&len);
for(k=st;k<(st+len);k++)
if(f[k]==0)
count++;
if(len==count)
{
for(j=st;j<(st+len);j++)
if(f[j]==0)
{
f[j]=1;
printf("%d\t%d\n",j,f[j]);
}
if(j!=(st+len-1))
printf("The file is allocated to disk\n");
}
else
printf("The file is not allocated \n");
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
getch();
}
```



```
Files Allocated are :
Enter starting block and length of files: 5
3
5       1
6       1
7       1
The file is allocated to disk
Do you want to enter more file(Yes - 1/No - 0)
```

35. 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 main()
{
int f[50], index[50],i, n, st, len, j, c, k, ind,count=0;
for(i=0;i<50;i++)
f[i]=0;
x:printf("Enter the index block: ");
scanf("%d",&ind);
if(f[ind]!=1)
{
printf("Enter no of blocks needed and no of files for the index %d on the disk : \n", ind);
scanf("%d",&n);
}
else
{
printf("%d index is already allocated \n",ind);
goto x;
}
y: count=0;
for(i=0;i<n;i++)
{
scanf("%d", &index[i]);
if(f[index[i]]==0)
count++;
}
if(count==n)
{
for(j=0;j<n;j++)
f[index[j]]=1;
printf("Allocated\n");
printf("File Indexed\n");
for(k=0;k<n;k++)
printf("%d-------->%d : %d\n",ind,index[k],f[index[k]]);
}
else
{
printf("File in the index is already allocated \n");
printf("Enter another file indexed");
goto y;
}
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
getch();
}
```

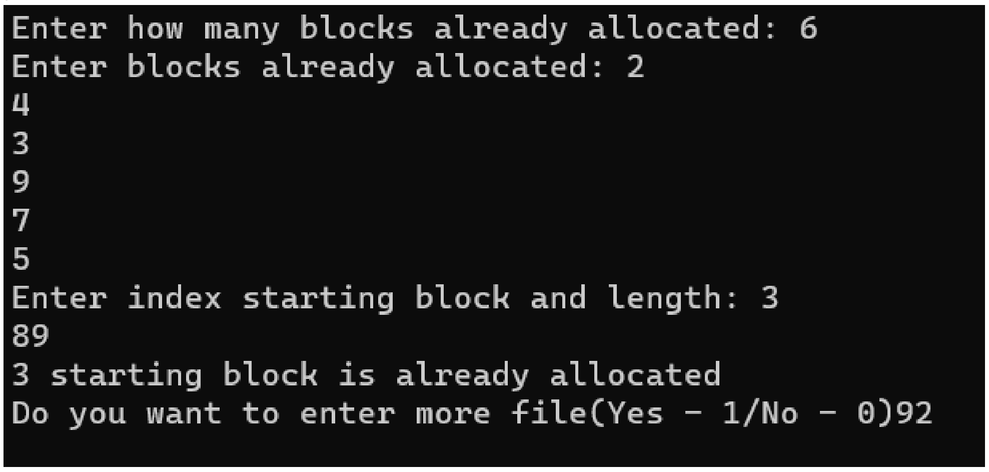


```
Enter the index block: 5
Enter no of blocks needed and no of files for the index 5 on the disk :
4
3
9
6
7
Allocated
File Indexed
5-------->3 : 1
5-------->9 : 1
5-------->6 : 1
5-------->7 : 1
Do you want to enter more file(Yes - 1/No - 0)0
```

36. 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>
main()
{
int f[50], p,i, st, len, j, c, k, a;

for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks already allocated: ");
scanf("%d",&p);
printf("Enter blocks already allocated: ");
for(i=0;i<p;i++)
{
```

```
scanf("%d",&a);
f[a]=1;
}
x: printf("Enter index starting block and length: ");
scanf("%d%d", &st,&len);
k=len;
if(f[st]==0)
{
for(j=st;j<(st+k);j++)
{
if(f[j]==0)
{
f[j]=1;
printf("%d-------->%d\n",j,f[j]);
}
else
{
printf("%d Block is already allocated \n",j);
k++;
}
}
}
else
printf("%d starting block is already allocated \n",st);
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
getch();
}
```

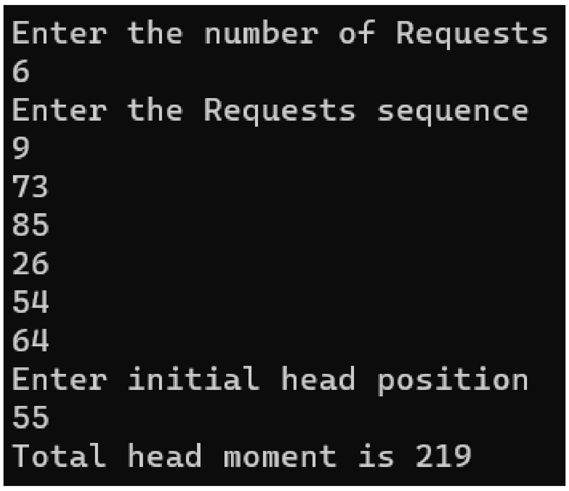


```
Enter how many blocks already allocated: 6
Enter blocks already allocated: 2
4
3
9
7
5
Enter index starting block and length: 3
89
3 starting block is already allocated
Do you want to enter more file(Yes - 1/No - 0)92
```

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

```
#include<stdio.h>
#include<stdlib.h>
int main()
```

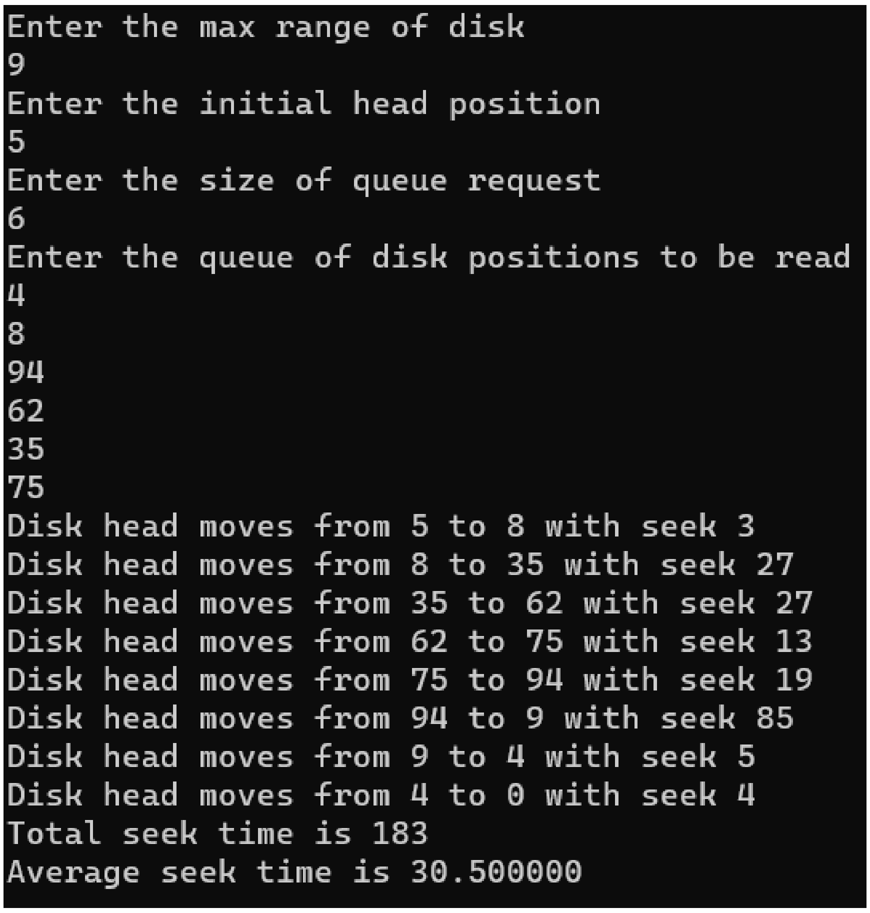
```
{
   int RQ[100],i,n,TotalHeadMoment=0,initial;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);
   for(i=0;i<n;i++)
   {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];
   }
   printf("Total head moment is %d",TotalHeadMoment);
   return 0;

}
```



```
Enter the number of Requests
6
Enter the Requests sequence
9
73
85
26
54
64
Enter initial head position
55
Total head moment is 219
```

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

```
#include <stdio.h>
#include <math.h>
int main()
{
   int queue[20], n, head, i, j, k, seek = 0, max, diff, temp, queue1[20],
   queue2[20], temp1 = 0, temp2 = 0;
   float avg;
   printf("Enter the max range of disk\n");
   scanf("%d", &max);
   printf("Enter the initial head position\n");
   scanf("%d", &head);
   printf("Enter the size of queue request\n");
   scanf("%d", &n);
   printf("Enter the queue of disk positions to be read\n");
   for (i = 1; i <= n; i++)
   {
      scanf("%d", &temp);
```

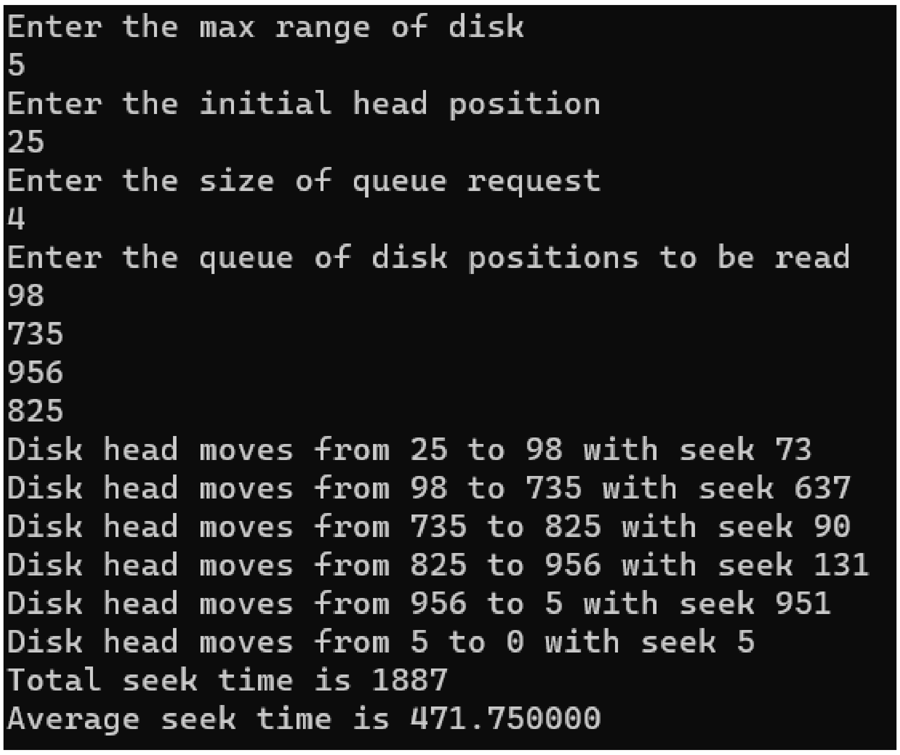
```
        if (temp >= head)
        {
            queue1[temp1] = temp;
            temp1++;
        }
        else
        {
            queue2[temp2] = temp;
            temp2++;
        }
    }
    for (i = 0; i < temp1 - 1; i++)
    {
        for (j = i + 1; j < temp1; j++)
        {
            if (queue1[i] > queue1[j])
            {
                temp = queue1[i];
                queue1[i] = queue1[j];
                queue1[j] = temp;
            }
        }
    }
    for (i = 0; i < temp2 - 1; i++)
    {
        for (j = i + 1; j < temp2; j++)
        {
            if (queue2[i] < queue2[j])
            {
                temp = queue2[i];
                queue2[i] = queue2[j];
                queue2[j] = temp;
            }
        }
    }
    for (i = 1, j = 0; j < temp1; i++, j++)
        queue[i] = queue1[j];
    queue[i] = max;
    for (i = temp1 + 2, j = 0; j < temp2; i++, j++)
        queue[i] = queue2[j];
    queue[i] = 0;
    queue[0] = head;
    for (j = 0; j <= n + 1; j++)
    {
        diff = abs(queue[j + 1] - queue[j]);
        seek += diff;
        printf("Disk head moves from %d to %d with seek %d\n", queue[j],
        queue[j + 1], diff);
    }
    printf("Total seek time is %d\n", seek);
    avg = seek / (float)n;
    printf("Average seek time is %f\n", avg);
    return 0;
}
```


```
Enter the max range of disk
9
Enter the initial head position
5
Enter the size of queue request
6
Enter the queue of disk positions to be read
4
8
94
62
35
75
Disk head moves from 5 to 8 with seek 3
Disk head moves from 8 to 35 with seek 27
Disk head moves from 35 to 62 with seek 27
Disk head moves from 62 to 75 with seek 13
Disk head moves from 75 to 94 with seek 19
Disk head moves from 94 to 9 with seek 85
Disk head moves from 9 to 4 with seek 5
Disk head moves from 4 to 0 with seek 4
Total seek time is 183
Average seek time is 30.500000
```

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

```
#include <stdio.h>
#include <math.h>
int main()
{
    int queue[20], n, head, i, j, k, seek = 0, max, diff, temp, queue1[20],
    queue2[20], temp1 = 0, temp2 = 0;
    float avg;
    printf("Enter the max range of disk\n");
    scanf("%d", &max);
    printf("Enter the initial head position\n");
    scanf("%d", &head);
    printf("Enter the size of queue request\n");
    scanf("%d", &n);
    printf("Enter the queue of disk positions to be read\n");
    for (i = 1; i <= n; i++)
    {
        scanf("%d", &temp);
        if (temp >= head)
    {
        queue1[temp1] = temp;
        temp1++;
    }
    else
    {
        queue2[temp2] = temp;
        temp2++;
    }
}
for (i = 0; i < temp1 - 1; i++)
{
    for (j = i + 1; j < temp1; j++)
    {
        if (queue1[i] > queue1[j])
        {
            temp = queue1[i];
            queue1[i] = queue1[j];
            queue1[j] = temp;
        }
    }
}
for (i = 0; i < temp2 - 1; i++)
{
    for (j = i + 1; j < temp2; j++)
    {
        if (queue2[i] > queue2[j])
        {
            temp = queue2[i];
            queue2[i] = queue2[j];
            queue2[j] = temp;
        }
    }
}
for (i = 1, j = 0; j < temp1; i++, j++)
queue[i] = queue1[j];
queue[i] = max;
queue[i + 1] = 0;
for (i = temp1 + 3, j = 0; j < temp2; i++, j++)
queue[i] = queue2[j];
queue[0] = head;
for (j = 0; j <= n + 1; j++)
{
diff = abs(queue[j + 1] - queue[j]);
seek += diff;
printf("Disk head moves from %d to %d with seek %d\n", queue[j],
queue[j + 1], diff);
}
```

```
    printf("Total seek time is %d\n", seek);
    avg = seek / (float)n;
    printf("Average seek time is %f\n", avg);
    return 0;
}
```

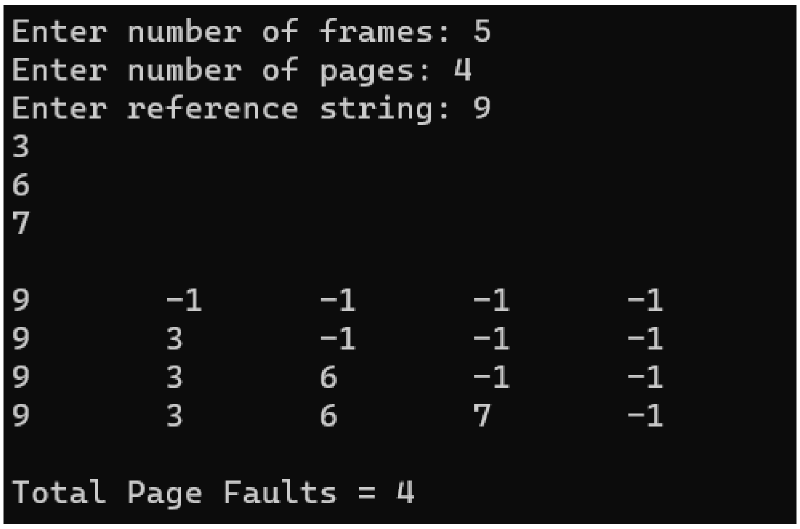


```
Enter the max range of disk
5
Enter the initial head position
25
Enter the size of queue request
4
Enter the queue of disk positions to be read
98
735
956
825
Disk head moves from 25 to 98 with seek 73
Disk head moves from 98 to 735 with seek 637
Disk head moves from 735 to 825 with seek 90
Disk head moves from 825 to 956 with seek 131
Disk head moves from 956 to 5 with seek 951
Disk head moves from 5 to 0 with seek 5
Total seek time is 1887
Average seek time is 471.750000
```

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

```
#include<stdio.h>
int main()
{
 FILE *fp;
 fp = fopen;
 if(!fp)
 {
 printf("Error in opening file\n");
 return 0;
 }
 printf("Position of the pointer : %ld\n",ftell(fp));
 char ch;
 while(fread(&ch,sizeof(ch),1,fp)==1)
 {
 printf("%c",ch);
 }
 printf("\nPosition of the pointer : %ld\n",ftell(fp));
 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)
 {
```

```
 printf("%c",ch);
 }
 fclose(fp);
 return 0;
}
```



```
Enter number of frames: 5
Enter number of pages: 4
Enter reference string: 9
3
6
7

9       -1      -1      -1      -1
9       3       -1      -1      -1
9       3       6       -1      -1
9       3       6       7       -1

Total Page Faults = 4
```