Experiment-1:

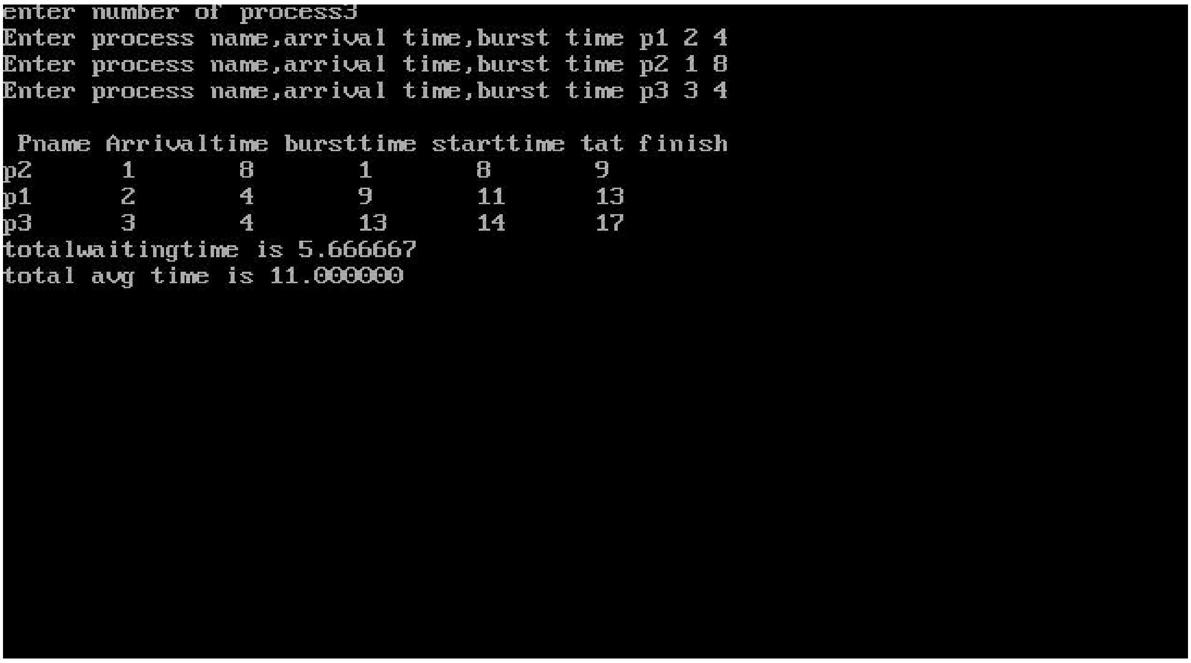
Simulate the following CPU scheduling algorithms:

1-(a) FCFS:

Code:

```
#include<stdio.h>
void main()
{
        char pn[10][10],t[10];
        int ar[10],bur[10],st[10],fin[10],tat[10],wt[10],i,j,n,p,q,temp;
        float totwt=0,tottat=0;
        clrscr();
        printf("enter number of process");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("Enter process name,arrival time,burst time ");
                scanf("%s %d %d",pn[i],&ar[i],&bur[i]);
        }
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                {
                        if(ar[i]<ar[j])
                        {
                                temp=ar[i];
                                ar[i]=ar[j];
                                ar[j]=temp;
                                temp=bur[i];
                                bur[i]=bur[j];
                                bur[j]=temp;
                                strcpy(t,pn[i]);
                                strcpy(pn[i],pn[j]);
                                strcpy(pn[j],t);
                        }
                }
        }
        for(i=0;i<n;i++)
        {
                if(i==0)
                {
                        st[i]=ar[i];
                        wt[i]=st[i]-ar[i];
                        fin[i]=st[i]+bur[i];
                tat[i]=fin[i]-ar[i];
                }
                else
                {
                        st[i]=fin[i-1];
                        wt[i]=st[i]-ar[i];
                        fin[i]=st[i]+bur[i];
                        tat[i]=fin[i]-ar[i];
                }
        }
        printf("\n Pname Arrivaltime bursttime starttime tat finish");
        for(i=0;i<n;i++)
        {
                printf("\n%s\t%d\t%d\t%d\t%d\t%d",pn[i],ar[i],bur[i],st[i],tat[i],fin[i]);
                totwt+=wt[i];
                tottat+=tat[i];
        }
        printf("\ntotalwaitingtime is %f",(totwt/n));
        printf("\ntotal avg time is %f",(tottat/n));
        getch();
        return;
}
```

Output:



```
enter number of process3
Enter process name,arrival time,burst time p1 2 4
Enter process name,arrival time,burst time p2 1 8
Enter process name,arrival time,burst time p3 3 4

 Pname Arrivaltime bursttime starttime tat finish
p2      1       8       1       8       9
p1      2       4       9       11      13
p3      3       4       13      14      17
totalwaitingtime is 5.666667
total avg time is 11.000000
```

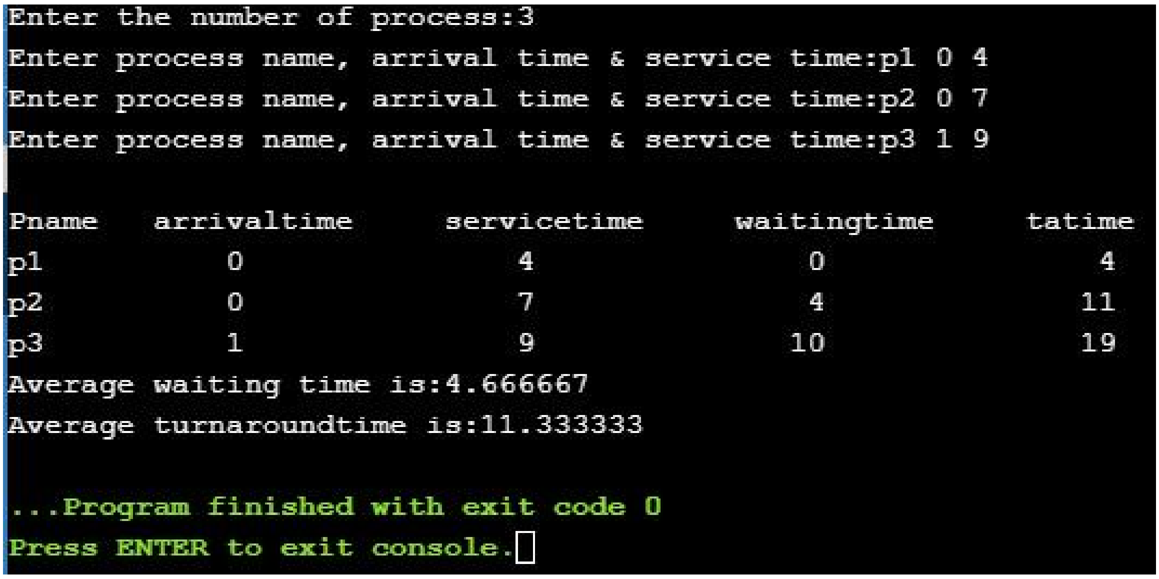
1-(b) SJF:

Code:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
        int et[20],at[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10];
        int totwt=0,totta=0;
        float awt,ata;
        char pn[10][10],t[10];
        printf("Enter the number of process:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
   {
        printf("Enter process name, arrival time & service time:");
        scanf("%s%d%d",pn[i],&at[i],&et[i]);
   }
        for(i=0;i<n;i++)
        {
           for(j=0;j<n;j++)
           {
```

```
                        if(et[i]<et[j])
                    {
                                temp=at[i];
                                at[i]=at[j];
                                at[j]=temp;
                                temp=et[i];
                                et[i]=et[j];
                                et[j]=temp;
                                strcpy(t,pn[i]);
                                strcpy(pn[i],pn[j]);
                                strcpy(pn[j],t);
                    }
            }
    }
            for(i=0;i<n;i++)
            {
                        if(i==0)
                                st[i]=at[i];
                        else
                                st[i]=ft[i-1];
                        wt[i]=st[i]-at[i];
                        ft[i]=st[i]+et[i];
                        ta[i]=ft[i]-at[i];
                        totwt+=wt[i];
                        totta+=ta[i];
            }
            awt=(float)totwt/n;
            ata=(float)totta/n;
            printf("\nPname\tarrivaltime\tservicetime\twaitingtime\ttatime");
            for(i=0;i<n;i++)
               printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],wt[i],ta[i]);
            printf("\nAverage waiting time is:%f",awt);
            printf("\nAverage turnaroundtime is:%f",ata);
            getch();
}
```

Output:



```
Enter the number of process:3
Enter process name, arrival time & service time:p1 0 4
Enter process name, arrival time & service time:p2 0 7
Enter process name, arrival time & service time:p3 1 9

Pname   arrivaltime     servicetime     waitingtime     tatime
p1          0               4               0               4
p2          0               7               4              11
p3          1               9              10              19
Average waiting time is:4.666667
Average turnaroundtime is:11.333333

...Program finished with exit code 0
Press ENTER to exit console.
```

Experiment-2:

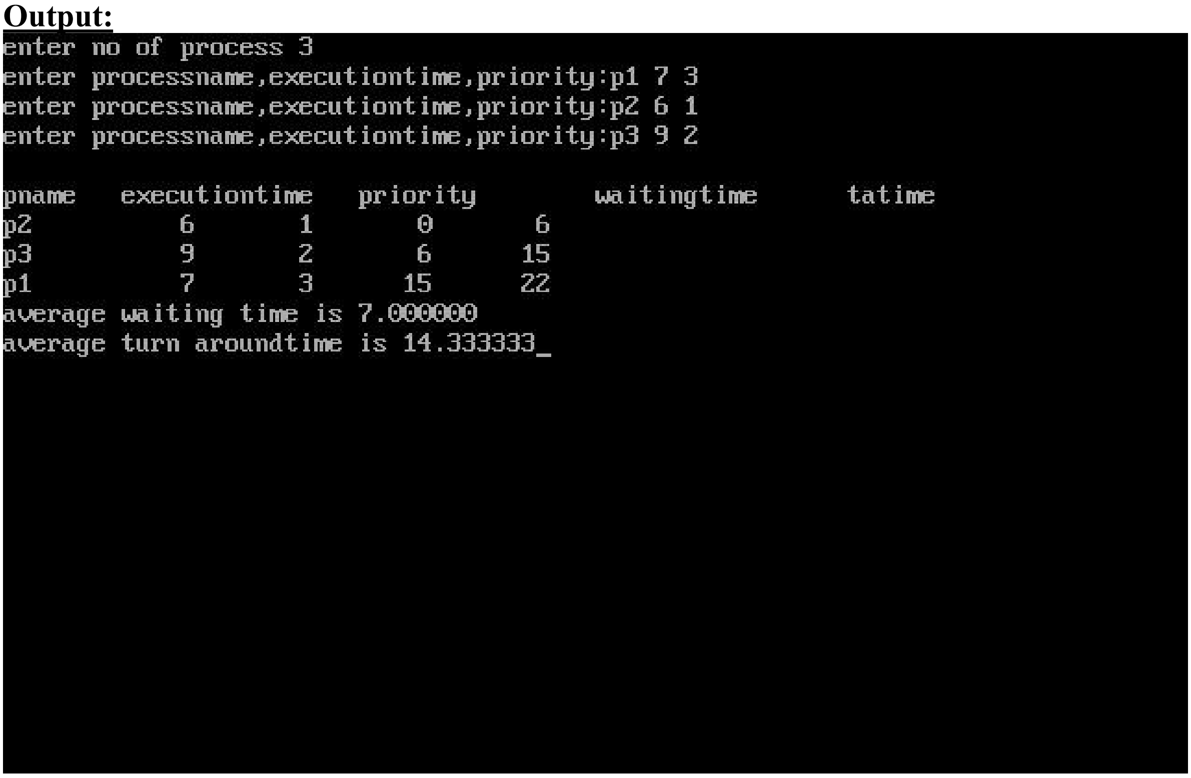
Simulate the following CPU scheduling algorithms:

2-(a) Priority:

Code:

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
void main()
{
        int et[10],temp,n,i,j,p[10],st[10],ft[10],wt[10],ta[10];
        int totwt=0,tota=0;
        float awt,ata;
        char pn[10][10],t[10];
        clrscr();
        printf("enter no of process");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("enter processname,executiontime,priority:");
                scanf("%s%d%d",&pn[i],&et[i],&p[i]);
        }
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                {
                        if(p[i]<p[j])
                        {
                                temp=p[i];
                                p[i]=p[j];
                                p[j]=temp;
                                temp=et[i];
                                et[i]=et[j];
                                et[j]=temp;
                                strcpy(t,pn[i]);
                                strcpy(pn[i],pn[j]);
                                strcpy(pn[j],t);
                        }
                }
        }
        for(i=0;i<n;i++)
        {
                if(i==0)
                {
                        st[i]=wt[i]=0;
                        ft[i]=st[i]+et[i];
                        ta[i]=ft[i];
                }
                else
                {
                        st[i]=ft[i-1];
                        wt[i]=st[i];
                        ft[i]=st[i]+et[i];
                        ta[i]=ft[i];
                }
                totwt+=wt[i];
                tota+=ta[i];
        }
        awt=(float)totwt/n;
        ata=(float)tota/n;
        printf("\npname\texecutiontime\tpriority\twaitingtime\t tatime");
        for(i=0;i<n;i++)
                printf("\n%s\t%5d\t%5d\t%5d\t%5d",pn[i],et[i],p[i],wt[i],ta[i]);
        printf("\naverage waiting time is %f",awt);
        printf("\naverage turn aroundtime is %f",ata);
        getch();
}
```

Output:



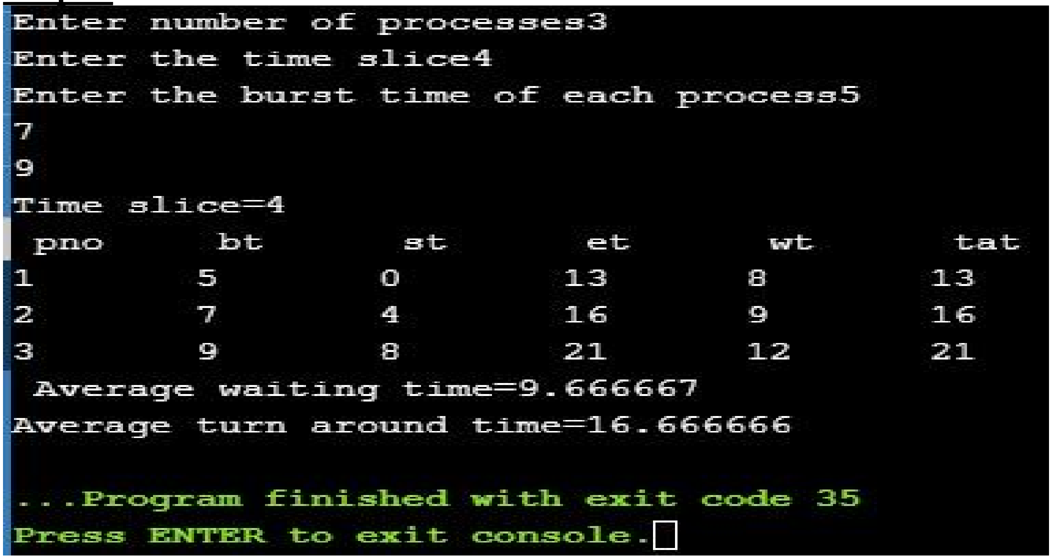
```
enter no of process 3
enter processname,executiontime,priority:p1 7 3
enter processname,executiontime,priority:p2 6 1
enter processname,executiontime,priority:p3 9 2

pname   executiontime   priority        waitingtime      tatime
p2          6       1       0       6
p3          9       2       6      15
p1          7       3      15      22
average waiting time is 7.000000
average turn aroundtime is 14.333333_
```

2-(b) Round Robin:

Code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int b[10], pno[10],ts,n,s[10],e[10],w[10],t[10],r[10];
        int i,c=0,x=0;
        float aw=0,at=0;
        printf("Enter number of processes");
        scanf("%d",&n);
        for(i=0;i<n;i++)
                pno[i]=i+1;
        printf("Enter the time slice");
        scanf("%d",&ts);
        printf("Enter the burst time of each process");
        for(i=0;i<n;i++)
                scanf("%d",&b[i]);
        s[0]=0;
        x=0;
        c=0;
        for(i=0;i<n;i++)
        {
                if(b[i]<ts)
           {
                e[i]=x+b[i];
                r[i]=0;
           }
                else
           {
                e[i]=ts+x;
                r[i]=b[i]-ts;
           }
                x=e[i];
                s[i+1]=e[i];
                t[i]=e[i];
                w[i]=s[i];
        }
        while(c>=0)
        {
                for(i=0;i<n;i++)
           {
                if(r[i]!=0)
                {
                        w[i]=w[i]+x-e[i];
                        if(r[i]<ts)
                {
                        e[i]=x+r[i];
                        r[i]=0;
                        }
                        else
                        {
                                e[i]=x+ts;
                                r[i]=r[i]-ts;
                        }
                        x=e[i];
                        t[i]=e[i];
                }
                if(r[i]!=0)
                        c++;
                }
                c--;
        }
        for(i=0;i<n;i++)
        {
                aw=aw+w[i];
                at=at+t[i];
        }
        aw=aw/n;
        at=at/n;
        printf("Time slice=%d",ts);
        printf("\n pno \t bt \t st \t et \t wt \t tat");
        for(i=0;i<n;i++)
                printf("\n%d\t%d\t%d\t%d\t%d\t%d",pno[i],b[i],s[i],e[i],w[i],t[i]);
        printf("\n Average waiting time=%f",aw);
        printf("\nAverage turn around time=%f",at);
}
```

Output:



```
Enter number of processes3
Enter the time slice4
Enter the burst time of each process5
7
9
Time slice=4
 pno     bt      st      et      wt      tat
1        5       0       13      8       13
2        7       4       16      9       16
3        9       8       21      12      21
 Average waiting time=9.666667
Average turn around time=16.666666

...Program finished with exit code 35
Press ENTER to exit console.
```

Experiment-3:

Multiprogramming-Memory management-Implementation of fork (), wait (), exec() and exit (), System calls

In Windows:

Code:

```
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
int main()
{
        pid_t p;
        p=fork();
        if(p==-1)
        {
                printf("Fork Error");
                exit(0);
        }
        else if(p==0)
        {
                int i;
                printf("Child PID is %d and PPID is %d\n",getpid(),getppid());
                for(i=1;i<6;i++)
                {
                        printf("Child i is %d\n",i);
                }
                _exit(0);
        }
        else
        {
                int i;
                printf("Parent PID is %d and PPID is %d\n",getpid(),getppid());
                pid_t p1=wait(0);
                printf("PID=%d child ended\n",p1);
                for(i=1;i<6;i++)
                {
                        printf("Parent i is %d\n",i);
                }
                exit(0);
        }
        return 0;
}
```

Output-1:

```
Parent PID is 20561 and PPID is 20523
Child PID is 20562 and PPID is 20561
Child i is 1
Child i is 2
Child i is 3
Child i is 4
Child i is 5
```

Output-2:

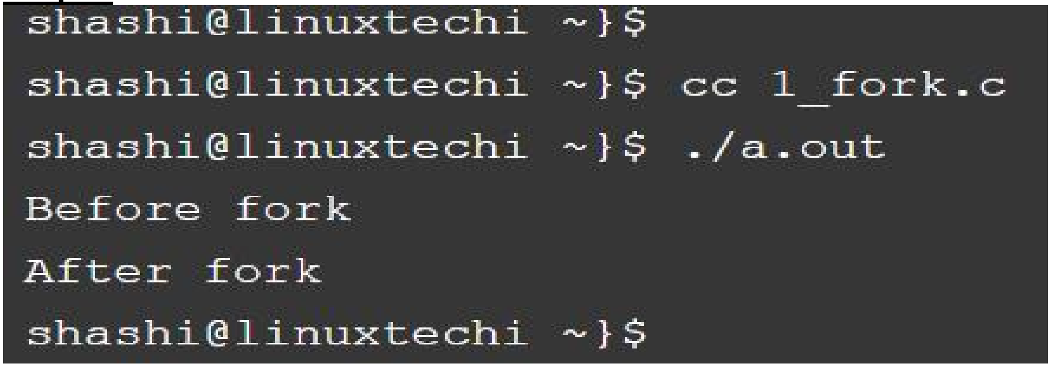
```
PID=20562 child ended
Parent i is 1
Parent i is 2
Parent i is 3
Parent i is 4
Parent i is 5
```

In PuTTY:

Code:

```
#include<stdio.h>
void main(int arc,char*ar[])
{
        int pid;
        char s[100];
        pid=fork();
        if(pid<0)
                printf("error");
        else if(pid>0)
        {
                wait(NULL);
                printf("\n Parent Process:\n");
                printf("\n\tParent Process id:%d\t\n",getpid());
                execlp("cat","cat",ar[1],(char*)0); error("can’t execute cat %s,",ar[1]);
        }
        else
        {
                printf("\nChild process:");
                printf("\n\tChildprocess parent id:\t %d",getppid());
                sprintf(s,"\n\tChild process id :\t%d",getpid());
                write(1,s,strlen(s));
                printf(" ");
                printf(" ");
                printf(" ");
                execvp(ar[2],&ar[2]);
                error("can’t execute %s",ar[2]);
        }
}
```

Output:



```
shashi@linuxtechi ~}$
shashi@linuxtechi ~}$ cc 1_fork.c
shashi@linuxtechi ~}$ ./a.out
Before fork
After fork
shashi@linuxtechi ~}$
```

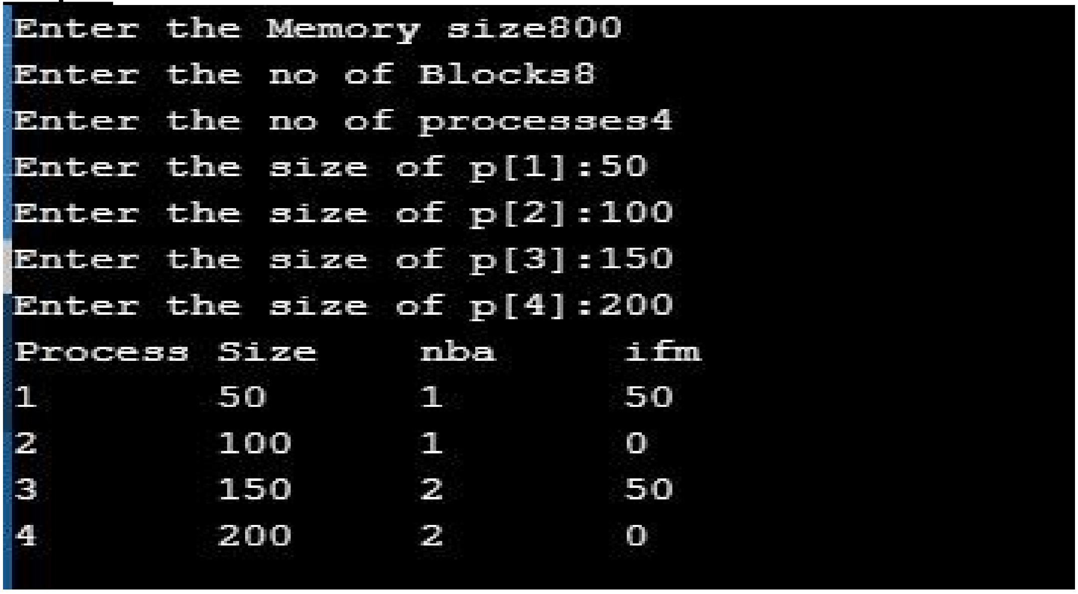
Experiment-4:

Simulate the Multiprogramming with a fixed number of tasks (MFT)

Code:

```
#include<stdio.h>
#include<math.h>
void main(){
        int np,nb,mm,bs,i,j,ps[100],nba[100],ifm[100],sb=0,flag=0;
        float x;
        printf("Enter the Memory size");
        scanf("%d",&mm);
        printf("Enter the no of Blocks");
        scanf("%d",&nb);
        printf("Enter the no of processes");
        scanf("%d",&np);
        bs=mm/nb;
        for(i=1;(i<=np)&&(sb<nb);i++){
                printf("Enter the size of p[%d]:",i);
                scanf("%d",&ps[i]);
                if(ps[i]<=bs)
                        nba[i]=1;
                else{
                        x=ps[i]/(float)bs;
                        nba[i]=(ceil)(x);
                }
                ifm[i]=nba[i]*bs-ps[i];
                sb=sb+nba[i];
                if(sb>nb){
                        i=i-1;
                        flag=1;
                }
        }
        j=i;
        printf("Process\tSize\tnba\tifm\n");
        for(i=1;i<j;i++)
                printf("%d\t%d\t%d\t%d\n",i,ps[i],nba[i],ifm[i]);
        if(flag==1)
                printf("Memory space is unavailable");
        getch();
}
```

Output:



```
Enter the Memory size800
Enter the no of Blocks8
Enter the no of processes4
Enter the size of p[1]:50
Enter the size of p[2]:100
Enter the size of p[3]:150
Enter the size of p[4]:200
Process Size    nba     ifm
1       50      1       50
2       100     1       0
3       150     2       50
4       200     2       0
```

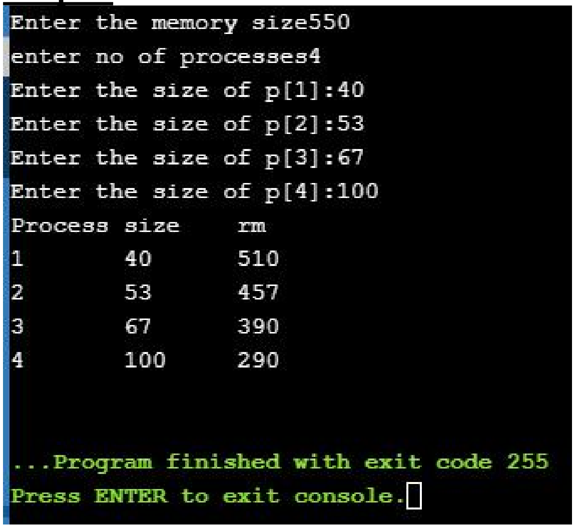
Experiment-5:

Simulate the Multiprogramming with a variable number of tasks (MVT)

Code:

```
#include<stdio.h>
void main()
{
        int mm,np,ps[100],rm[100],am=0,flag=0,i,j;
        printf("Enter the memory size");
        scanf("%d",&mm);
        printf("enter no of processes");
        scanf("%d",&np);
        for(i=0;(i<np)&&(am<mm);i++)
        {
                printf("Enter the size of p[%d]:",i+1);
                scanf("%d",&ps[i]);
                am=am+ps[i];
                if(am>=mm)
                {
                        flag=1;
                        break;
                }
                rm[i]=mm-am;
        }
        j=i;
        printf("Process\tsize\trm\n");
        for(i=0;i<j;i++)
                printf("%d\t%d\t%d\n",i+1,ps[i],rm[i]);
        if(flag==1)
                printf("memory is unavailable");
        getch();
}
```

Output:



```
Enter the memory size550
enter no of processes4
Enter the size of p[1]:40
Enter the size of p[2]:53
Enter the size of p[3]:67
Enter the size of p[4]:100
Process size    rm
1       40      510
2       53      457
3       67      390
4       100     290


...Program finished with exit code 255
Press ENTER to exit console.
```

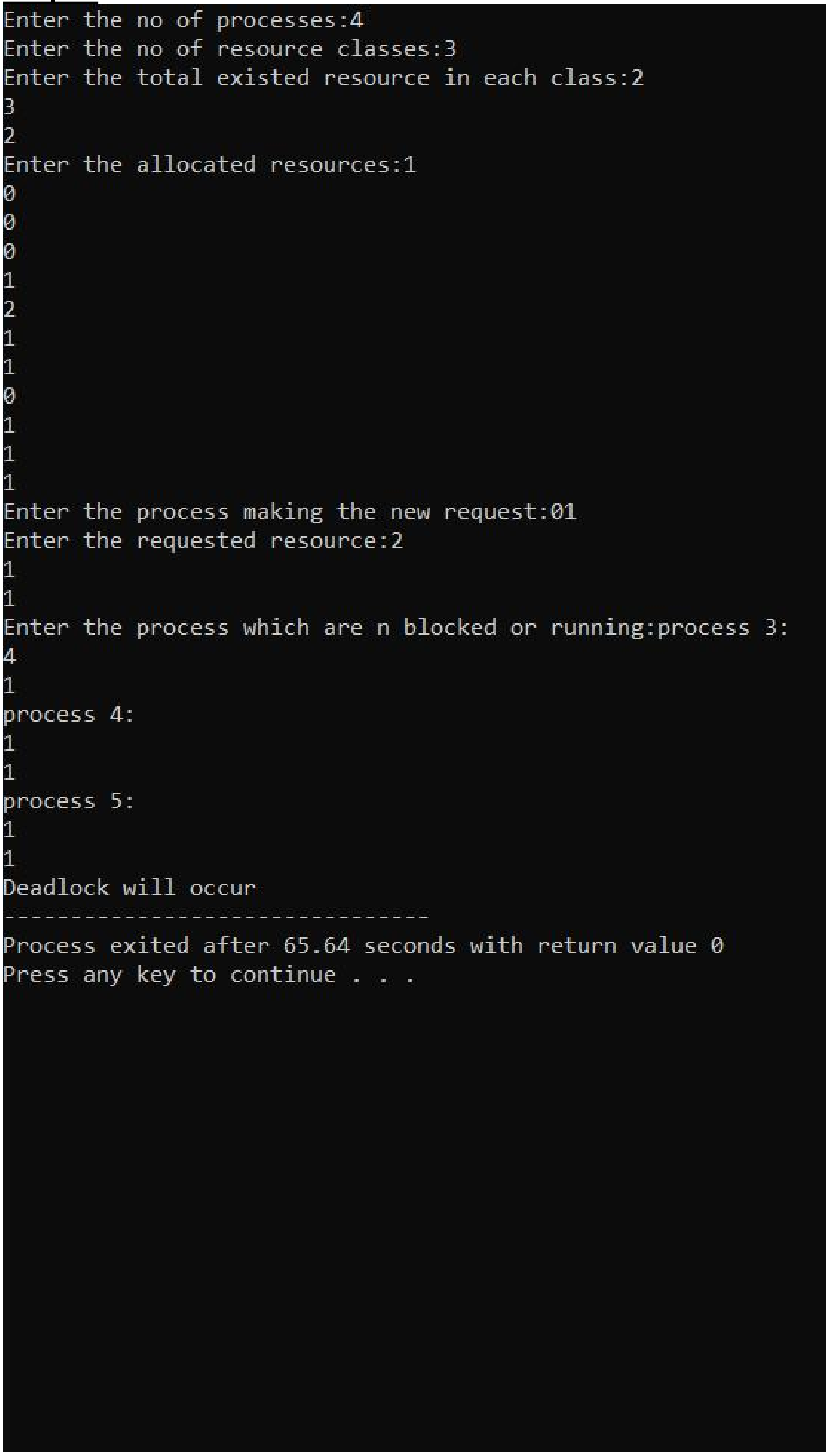
Experiment-6:

Simulate Bankers Algorithm for Dead Lock Avoidance

Code:

```
#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 of resource 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();
}
```

Output:



```
Enter the no of processes:4
Enter the no of resource classes:3
Enter the total existed resource in each class:2
3
2
Enter the allocated resources:1
0
0
0
1
2
1
1
0
1
1
1
Enter the process making the new request:01
Enter the requested resource:2
1
1
Enter the process which are n blocked or running:process 3:
4
1
process 4:
1
1
process 5:
1
1
Deadlock will occur
--------------------------------
Process exited after 65.64 seconds with return value 0
Press any key to continue . . .
```

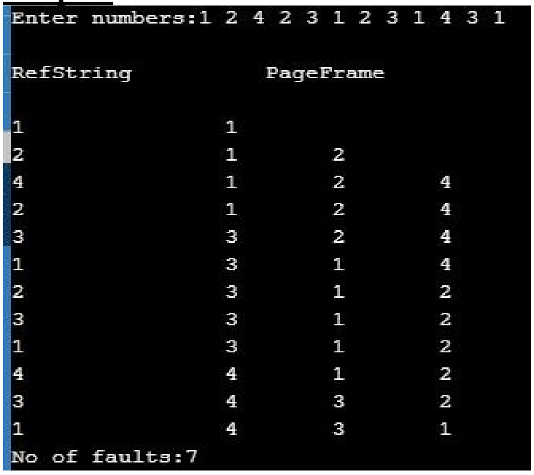
Experiment-7:

Simulate the FIFO page replacement algorithm

Code:

```
#include<stdio.h>
#include<conio.h>
void main(){
        int a[5],b[20],p=0,q=0,m=0,h,k,i,q1=1,j,u;
        char f='F';
        printf("Enter numbers:");
        for(i=0;i<12;i++)
        scanf("%d",&b[i]);
        printf("\nRefString         PageFrame\n");
        for(i=0;i<12;i++){
                if(p==0){
                        if(q>=3)
                                q=0;
                        a[q]=b[i];
                        q++;
                        if(q1<3)
                                q1=q;}
                printf("\n%d",b[i]);
                printf("\t");
                for(h=0;h<q1;h++)
                        printf("\t%d",a[h]);
                if((p==0)&&(q1==3))
                        m++;
                p=0;
                for(k=0;k<q-1;k++){
                        if(b[i+1]==a[k])
                                p=1;}
        }
        printf("\nNo of faults:%d",m);
        getch();
}
```

Output:



```
Enter numbers:1 2 4 2 3 1 2 3 1 4 3 1

RefString          PageFrame

1               1
2               1       2
4               1       2       4
2               1       2       4
3               3       2       4
1               3       1       4
2               3       1       2
3               3       1       2
1               3       1       2
4               4       1       2
3               4       3       2
1               4       3       1
No of faults:7
```

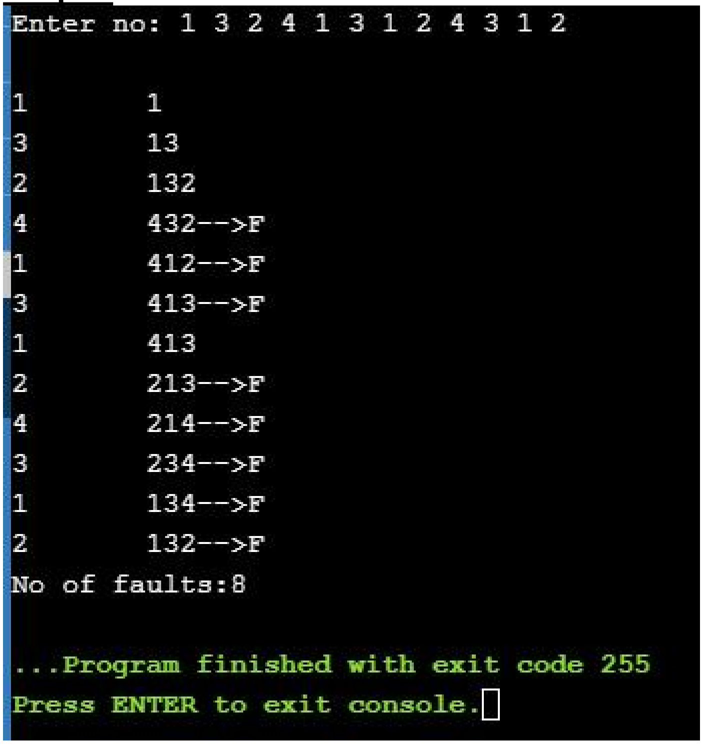
Experiment-8:

Simulate the LRU page replacement algorithm

Code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int g=0,a[5],b[20],p=0,q=0,m=0,h,k,i,q1=1,j,u;
        char f='F';
        printf("Enter no: ");
        for(i=0;i<12;i++)
                scanf("%d",&b[i]);
        for(i=0;i<12;i++)
        {
                if(p==0)
                {
                        if(q>=3)
                                q=0;
                        a[q]=b[i];
                        q++;
                        if(q1<3)
                        {
                                q1=q;
                                g=1;
                        }
                }
                printf("\n%d",b[i]);
                printf("\t");
                for(h=0;h<q1;h++)
                        printf("%d",a[h]);
                if((p==0)&&(q1==3)&&(g!=1))
                {
                        printf("-->%c",f);
                        m++;
                }
                p=0;
                g=0;
                if(q1==3)
                {
                        for(k=0;k<q-1;k++)
                        {
                                if(b[i+1]==a[k])
                                        p=1;
                        }
                        for(j=0;j<q1;j++)
                        {
                                u=0;
                                k=i;
                while(k>(i-2)&&(k>=0))
                {
                    if(b[k]==a[j])
                    u++;
                    k--;
                }
                if(u==0)
                    q=j;
            }
        }
        else
        {
            for(k=0;k<q;k++)
            {
                if(b[i+1]==a[k])
                    p=1;
            }
        }
    }
    printf("\nNo of faults:%d",m);
    getch();
}
```

Output:



```
Enter no: 1 3 2 4 1 3 1 2 4 3 1 2

1       1
3       13
2       132
4       432-->F
1       412-->F
3       413-->F
1       413
2       213-->F
4       214-->F
3       234-->F
1       134-->F
2       132-->F
No of faults:8

...Program finished with exit code 255
Press ENTER to exit console.
```

Experiment-9:

Simulate the following File allocation strategies

(a) Sequenced

Code:

```
#include<stdio.h>
#include<conio.h>
void main(){
        int n,i,j,b[20],sb[20],t[20],x,c[20][20];
        clrscr();
        printf("Enter no.of files:");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                printf("Enter no. of blocks occupied by file%d",i+1);
                scanf("%d",&b[i]);
                printf("Enter the starting block of file%d",i+1);
                scanf("%d",&sb[i]);
                t[i]=sb[i];
                for(j=0;j<b[i];j++)
                        c[i][j]=sb[i]++;
        }
        printf("Filename\tStart block\tlength\n");
        for(i=0;i<n;i++)
                printf("%d\t  %d \t%d\n",i+1,t[i],b[i]);
        printf("Enter file name:");
        scanf("%d",&x);
        printf("File name is:%d",x);
        printf("length is:%d",b[x-1]);
        printf("blocks occupied:");
        for(i=0;i<b[x-1];i++)
                printf("%4d",c[x-1][i]);
        getch();
}
```

Output:

```
Enter no.of files: 2
Enter no. of blocks occupied by file1 4
Enter the starting block of file1 2
 Enter no. of blocks occupied by file2 10
Enter the starting block of file2 5
Filename        Start block     length
 1                   2              4
2                    5              10
Enter file name: rajesh
 File name is:12803 length is:0blocks occupied
```

(b) Indexed

Code:

```
#include<stdio.h>
#include<conio.h>
void main(){
        int n,m[20],i,j,sb[20],s[20],b[20][20],x;
        clrscr();
        printf("Enter no. of files:");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                printf("Enter starting block and size of file%d:",i+1);
                scanf("%d%d",&sb[i],&s[i]);
                printf("Enter blocks occupied by file%d:",i+1);
                scanf("%d",&m[i]);
                printf("enter blocks of file%d:",i+1);
                for(j=0;j<m[i];j++)
                        scanf("%d",&b[i][j]);
        }
        printf("\nFile\t index\tlength\n");
        for(i=0;i<n;i++)
                printf("%d\t%d\t%d\n",i+1,sb[i],m[i]);
        printf("\nEnter file name:");
        scanf("%d",&x);
        printf("file name is:%d\n",x);
        i=x-1;
        printf("Index is:%d",sb[i]);
        printf("Block occupied are:");
        for(j=0;j<m[i];j++)
                printf("%3d",b[i][j]);
        getch();
}
```

Output:

```
Enter no. of files:2
 Enter starting block and size of file1: 2   5
Enter blocks occupied by file1:10
enter blocks of file1:3
2 5  4 6 7  2  6 4 7
Enter starting block and size of file2: 3   4
Enter blocks occupied by file2:5
enter blocks of file2: 2  3   4  5   6
File     index  length
 1       2       10
2       3       5
Enter file name: venkat
file name is:12803
Index is:0Block occupied are:
```

(c) Linked

Code:

```
#include<stdio.h>
#include<conio.h>
struct file{
        char fname[10];
        int start,size,block[10];
}f[10];
void main(){
        int i,j,n;
        clrscr();
        printf("Enter no. of files:");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                printf("Enter file name:");
                scanf("%s",&f[i].fname);
                printf("Enter starting block:");
                scanf("%d",&f[i].start);
                f[i].block[0]=f[i].start;
                printf("Enter no.of blocks:");
                scanf("%d",&f[i].size);
                printf("Enter block numbers:");
                for(j=1;j<=f[i].size;j++)
                        scanf("%d",&f[i].block[j]);
        }
        printf("File\tstart\tsize\tblock\n");
        for(i=0;i<n;i++){
                printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);
                for(j=1;j<=f[i].size-1;j++)
                        printf("%d--->",f[i].block[j]);
                printf("%d",f[i].block[j]);
                printf("\n");
        }
        getch();
}
```

Output:

```
Enter no. of files:2
Enter file name:venkat
Enter starting block:20
Enter no.of blocks:6
Enter block numbers: 4
12
15
45
32
25
Enter file name:rajesh
Enter starting block:12
Enter no.of blocks:5
Enter block numbers:6
5
4
3
2
File    start   size    block
venkat  20      6       4--->12--->15--->45--->32--->25
rajesh  12      5       6--->5--->4--->3--->2
```

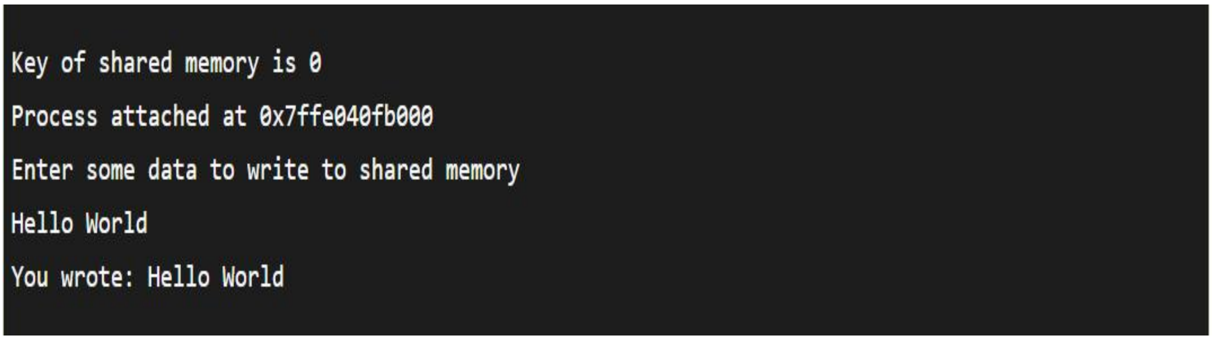
Experiment-10:

Write a C program that illustrates two processes communicating using shared memory

Process-1:

```
#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);
}
```

Output:

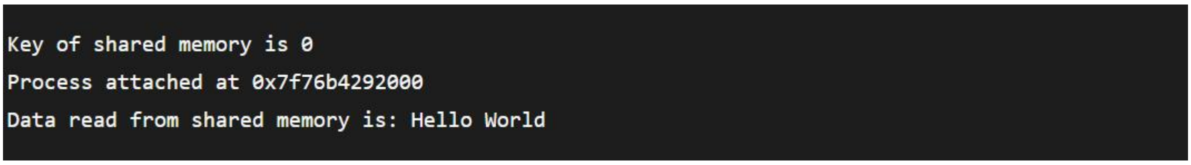


```
Key of shared memory is 0
Process attached at 0x7ffe040fb000
Enter some data to write to shared memory
Hello World
You wrote: Hello World
```

Process-2:

```
#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);
        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);
        printf("Data read from shared memory is : %s\n",(char *)shared_memory);
}
```

Output:



```
Key of shared memory is 0
Process attached at 0x7f76b4292000
Data read from shared memory is: Hello World
```

Experiment-11:

Write C program to create a thread using pthreads library and let it run its function.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
void *mythread(void *vargp)
{
        sleep(1);
        printf("welcome to AEC CSE\n");
        return NULL;
}
int main()
{
        pthread_t tid;
        printf("before thread\n");
        pthread_create(&tid,NULL,mythread,NULL);
        pthread_join(tid,NULL);
        exit(0);
}
```

Output:

```
Welcome to AEC CSE
```

Experiment-12:

Write a C program to illustrate concurrent execution of threads using pthreads library.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
void *mythread1(void *vargp){
        int i;
        printf("thread1\n");
        for(i=1;i<=10;i++)
        printf("i=%d\n",i);
        printf("exit from thread1\n");
        return NULL;
}
void *mythread2(void *vargp){
        int j;
        printf("thread2 \n");
        for(j=1;j<=10;j++)
        printf("j=%d\n",j);
        printf("Exit from thread2\n");
        return NULL;
}
int main(){
        pthread_t tid;
        printf("before thread\n");
        pthread_create(&tid,NULL,mythread1,NULL);
        pthread_create(&tid,NULL,mythread2,NULL);
        pthread_join(tid,NULL);
        pthread_join(tid,NULL);
        exit(0);
}
```

Output:

```
$ cc w8.c – l pthread
$./a.out
thread1
i=1
i=2;
i=3
thread2
j=1
j=2
j=3
j=4
j=5
j=6
j=7
j=8
i=4
i=5
i=6
i=7
i=8
i=9
i=10
exit from thread1
j=9
j=10
exit from thread2
```

Augmented Experiments:

Experiment-14:

Simulate Best-Fit contiguous memory allocation technique

Code:

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
        int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
        static int bf[max],ff[max];
        clrscr();
        printf("\n\tMemory Management Scheme – Best Fit");
        printf("\nEnter the number of blocks:"); scanf("%d",&nb);
        printf("Enter the number of files:"); scanf("%d",&nf);
        printf("\nEnter the size of the blocks:-\n");
        for(i=1;i<=nb;i++)
                printf("Block %d:",i);
        scanf("%d",&b[i]);
        printf("Enter the size of the files :-\n");
        for(i=1;i<=nf;i++){
                printf("File %d:",i);
                scanf("%d",&f[i]);
        }
        for(i=1;i<=nf;i++){
                for(j=1;j<=nb;j++){
                        if(bf[j]!=1){
                                temp=b[j]-f[i];
                                if(temp>=0)
                                if(lowest>temp)
                                {
                                ff[i]=j;
                                lowest=temp;
                                }
                        }
                }
                frag[i]=lowest;
                bf[ff[i]]=1;
                lowest=10000;
        }
        printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
        for(i=1;i<=nf && ff[i]!=0;i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
        getch();
}
```

Output:

```
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    Fragment
     1            1          2          2              1
2            4          1          5              1
```

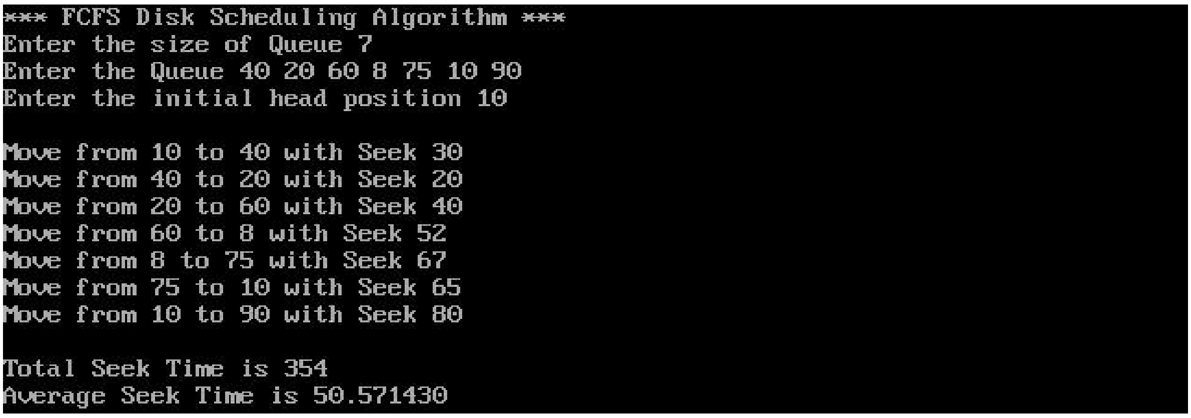
Experiment-15:

Simulate FCFS Disk Scheduling algorithm

Code:

```
#include<stdio.h>
void main()
{
        int queue[100],n,head,i,j,k,seek=0,diff;
        float avg;
        clrscr();
        printf("*** FCFS Disk Scheduling Algorithm ***\n");
        printf("Enter the size of Queue\t");
        scanf("%d",&n);
        printf("Enter the Queue\t");
        for(i=1;i<=n;i++)
                scanf("%d",&queue[i]);
        printf("Enter the initial head position\t");
        scanf("%d",&head);
        queue[0]=head;
        printf("\n");
        for(j=0;j<=n-1;j++){
                diff=abs(queue[j+1]-queue[j]);
                seek+=diff;
                printf("Move from %d to %d with Seek %d\n",queue[j],queue[j+1],diff);
        }
        printf("\nTotal Seek Time is %d\t",seek);
        avg=seek/(float)n;
        printf("\nAverage Seek Time is %f\t",avg);
        getch();
}
```

Output:



```
*** FCFS Disk Scheduling Algorithm ***
Enter the size of Queue 7
Enter the Queue 40 20 60 8 75 10 90
Enter the initial head position 10

Move from 10 to 40 with Seek 30
Move from 40 to 20 with Seek 20
Move from 20 to 60 with Seek 40
Move from 60 to 8 with Seek 52
Move from 8 to 75 with Seek 67
Move from 75 to 10 with Seek 65
Move from 10 to 90 with Seek 80

Total Seek Time is 354
Average Seek Time is 50.571430
```