UNIVERSITY COLLEGE OF ENGINEERING

(**OSMANIA UNIVERSITY**)

CERTIFICATE

This is to certify that Mr. /Miss _____

is a student of *MCA* _____ year_____ Semester bearing Hall Ticket No_____ has done the Practical Lab Record in Operating System Lab during the academic year 2023-24 .

INTERNAL EXAMINAR

EXTERNAL EXAMINAR

HEAD OF THE DEPARTMENT

OPERATING SYSTEMS LAB RECORD

INDEX

Contd.)

29 a) Shell Programs

1. Print Multiplication table of a given no. using all loops
2. Perform all arithmetic operations
3. Print the type of file……..
4. Rename all files whose names end with .c as .old……..
5. Display the no. of lines in each of text file in a given dir…

30a) AWK Programs

1 Find the total ……

……

2. Findsales……
……
3. Find department sales……..
4. Find the list of items……
5. Find the larger value……..

1.Program using system calls (fork, wait, exec, exit)

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.
h>
#include<sys/wait.h
>
#include<unistd.h>
int main(int argc, char **argv)
{
    pid_t pid;
    pid =
    fork();
    if(pid==0)
    {
        printf("It is the child process..with pid
        %d\n",getpid()); char *args[]={"./hello",NULL};
        printf("******Executing Another Program******\n");
        execv(args[0],args);
        printf("*******Ending********\n");
        exit(0);
    }
    else if(pid > 0)
    {
        printf("It is the parent process..with pid
        %d\n",getpid()); int status;
        wait(&status);
```

```
            printf("Child
            reaped\n");
        }
        else
        {
            printf("Error in
            forking..\n");
            exit(EXIT_FAILURE);
        }
        return 0;
}
```

//hello.c

```
#include<stdio.h
> int main(){
        printf("Hello
        World\n"); return 0;
}
```

OUTPUT:

```
It is the parent process..with pid
2236 It is the child process..with
pid 2237
******Executing Another Program******
Hello World
Child reaped
```

2.Program using read, write, open, close system calls.

```
#include<stdio.h>
#include<string.h
>
#include<fcntl.h>
#include<stdlib.h
>
#include<unistd.h
> int main(){
      int fd1,fd2;
      fd1=open("hello.txt",O_RDON
      LY); if(fd1==-1){
        perror("Error")
        ; exit(1);
      }
      char c[50];
      int
      r=read(fd1,c,50);
      c[r]='\0';
      printf("Read %d bytes\n",r);
      printf("Read Contents
      are\n%s",c); close(fd1);
      fd2=open("abc.txt",O_WRONLY|O_CR
      EAT); if(fd2<0)
            exit(1);
      int w=write(fd2,"System Calls\n",strlen("System
      Calls\n")); printf("Written %d bytes\n",w);
```

```
    close(fd2);
    return 0;
}
```

OUTPUT:

Read 12 bytes

Read Contents

are Hello World

Written 13 bytes

3. FCFS(First Come First Served) Scheduling Algorithm

```
#include<stdio.h
> int main(){
        int n,temp;
        printf("Enter no of process:
        "); scanf("%d",&n);
        int p[n],a[n],b[n],ct[n],tat[n],wt[n];
        int i,j;
        for(i=0;i<n;i++){
                printf("Enter process
                ID:"); scanf("%d",&p[i]);
                printf("Enter arrival
                time:"); scanf("%d",&a[i]);
                printf("Enter burst time:");
                scanf("%d",&b[i]);
        }
        for(i=0;i<n-1;i++){
                for(j=1;j<n;j++){
                        if(a[i]>a[j]){
                                temp=a[i]
                                ; a[i]=a[j];
                                a[j]=temp
                                ;
                                temp=p[i]
                                ; p[i]=p[j];
                                p[j]=temp
                                ;
                                temp=b[i]
```

```
                    ;
                    b[i]=b[j];
                    b[j]=temp
                    ;
            }
}
}
    int sum=a[0];
    float
    t=0.0,w=0.0;
    for(i=0;i<n;i++){
    if(a[i]<=sum)
            sum+=b[i];
     else
            sum=a[i]+b[i
     ]; ct[i]=sum;
     tat[i]=ct[i]-a[i];
     wt[i]=tat[i]-b[i];
     t+=tat[i];
     w+=wt[i];
    }
    t=t/n;
    w=w/n;
    printf("PID\tAT\tBT\tCT\tTAT\tWT\n"
    ); for(i=0;i<n;i++)
          printf("%d
    \t%d\t%d\t%d\t%d\t%d\n",p[i],a[i],b[i],ct[i],tat[i],wt[i]); printf("Average
    Waiting Time=%.2f\n",w);
    printf("Average Turn Around
    Time=%.2f",t); return 0;
```

}

INPUT:

Enter no of process:

3 Enter process

ID:1 Enter arrival

time:0 Enter burst

time:5 Enter

process ID:2 Enter

arrival time:3 Enter

burst time:9 Enter

process ID:3 Enter

arrival time:6 Enter

burst time:6

OUTPUT:

PID	AT	BT	CT	TAT	WT
1	0	5	5	5	0
2	3	9	14	11	2
3	6	6	20	14	8

Average Waiting Time=3.33

Average Turn around

Time=10.00

4. SJF(Shortest Job First) Scheduling Algorithm

```
#include<stdio.h
> int main(){
        int n,temp;
        printf("Enter no of process:
        "); scanf("%d",&n);
        int p[n],a[n],b[n],ct[n],tat[n],wt[n],r[n];
        int i,j;
        for(i=0;i<n;i++){
                printf("Enter process
                ID:"); scanf("%d",&p[i]);
                printf("Enter arrival
                time:"); scanf("%d",&a[i]);
                printf("Enter burst time:");
                scanf("%d",&b[i]);
                r[i]=999;
        }
        int
        order[n];
        int start=0;
   int t=0;
   while(1)
   {
        int f=0;
        for(i=0;i<n;i++
        ){ if(r[i]!=0)
            f=1;
         }
```

```
        if(f==0)
          break
          ;
        int min=999;
        int
        num,flag=0;
        for(j=0;j<n;j++
        )
                if(r[j]==999 &&
                   a[j]<=t) r[j]=b[j];
        for(j=0;j<n;j+
            +)
            if(r[j]!=999)
                flag=1
        ;
        if(flag==0){
              t++;
              continue;
       }

        for(j=0;j<n;j++){

                if(r[j]!=0&&r[j]<min){
                  min=r[j];
                  num=j;

                 }
        }
        order[start++]=nu
        m; t+=r[num];
        r[num]=0;
        ct[num]=t;
}
        int sum=0;
```

```
        float
        tt=0.0,w=0.0;
        for(i=0;i<n;i++){
                tat[i]=ct[i]-a[i];
                wt[i]=tat[i]-b[i];
                tt+=tat[i];
                w+=wt[i];
        }
        tt=tt/n;
        w=w/n;
        printf("PID\tAT\tBT\tCT\tTAT\tWT\n"
        ); for(i=0;i<n;i++){
         int k=order[i];
        printf("%d\t%d\t%d\t%d\t%d\t%d\n",p[k],a[k],b[k],ct[k],tat[k],wt[k]
        );
        }
         printf("Average Waiting
        Time=%.2f\n",w); printf("Average Turn
        Around Time=%.2f",tt); return 0;
}
```

INPUT:

Enter no of process:

4 Enter process

ID:1 Enter arrival

time:2 Enter burst

time:3 Enter

process ID:2 Enter

arrival time:0 Enter

burst time:4

Enter process

ID:3 Enter arrival

time:4 Enter burst

time:2 Enter

process ID:4

Enter arrival

time:5 Enter burst

time:4

OUTPUT:

PID	AT	BT	CT	TAT	WT
2	0	4	4	4	0
3	4	2	6	2	0
1	2	3	9	7	4
4	5	4	13	8	4

Average Waiting Time=2.00

Average Turn Around

Time=5.25

5. SRTF(Shortest Remaining Time First) Scheduling Algorithm

```
#include<stdio.h
> int main(){
        int n,temp;
        printf("Enter no of process:
        "); scanf("%d",&n);
        int p[n],a[n],b[n],ct[n],tat[n],wt[n],r[n];
        int i,j;
        for(i=0;i<n;i++){
                printf("Enter process
                ID:"); scanf("%d",&p[i]);
                printf("Enter arrival
                time:"); scanf("%d",&a[i]);
                printf("Enter burst time:");
                scanf("%d",&b[i]);
                r[i]=999;
        }
        int max=-1;
        for(i=0;i<n;i+
        +)
              if(a[i]>max)
                 max=a[i];
       int t=0;
        while(1)
        {
           int f=0;
```

```
    for(i=0;i<n;i++
     ){ if(r[i]!=0)
         f=1;
     }
    if(f==0) break;
    int min=999;
    int
    num,flag=0;
    for(j=0;j<n;j++
    ){
              if(r[j]==999 && a[j]<=t){
                 r[j]=b[j];
              }
    }
    for(j=0;j<n;j+
        +)
        if(r[j]!=999)
            flag=1
; if(flag==0){
   t++;
   continue;
}
    for(j=0;j<n;j++){
              if(r[j]!=0&&r[j]<min){
                min=r[j];
                num=j;
               }
    }
    if(t<max){
```

```
            r[num]-=1;
        t++;
       }
      else{
        t+=r[num]
        ;
        r[num]=0;
      }
      printf("%d--
      >",num+1);
      ct[num]=t;
}
      int sum=0;
      float
      tt=0.0,w=0.0;
      for(i=0;i<n;i++){
            tat[i]=ct[i]-a[i];
            wt[i]=tat[i]-b[i];
            tt+=tat[i];
            w+=wt[i];
      }
      tt=tt/n;
      w=w/n;
      printf("\nPID\tAT\tBT\tCT\tTAT\tWT\n"
      ); for(i=0;i<n;i++){
      printf("%d\t%d\t%d\t%d\t%d\t%d\n",p[i],a[i],b[i],ct[i],tat[i],wt[i]);
      }
      printf("Average Waiting Time=%.2f\n",w);
      printf("Average Turn Around
      Time=%.2f",tt);
```

```
    return 0;
}
```

INPUT:

Enter no of process:

4 Enter process

ID:1 Enter arrival

time:1 Enter burst

time:4 Enter

process ID:2 Enter

arrival time:2 Enter

burst time:4 Enter

process ID:3 Enter

arrival time:3 Enter

burst time:5 Enter

process ID:4 Enter

arrival time:4 Enter

burst time:8

OUTPUT:

1-->1-->1-->1-->2-->3-->4-->

PID	AT	BT	CT	TAT	WT
1	1	4	5	4	0
2	2	4	9	7	3
3	3	5	14	11	6
4	4	8	22	18	10

Average Waiting Time=4.75
Average Turn Around Time=10.00

6. Round Robin Algorithm

```
#include<stdio.h

> int q[10];

int f=-1,r=-

1; int

main(){

        int n,temp;

        printf("Enter no of process:

        "); scanf("%d",&n);

        int

        p[n],a[n],b[n],ct[n],tat[n],wt[n],b1[n];

        int visited[n];

        int i,j;

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

        ){

                visited[i]=0;

                printf("Enter process
                ID:"); scanf("%d",&p[i]);
                printf("Enter arrival
                time:"); scanf("%d",&a[i]);
                printf("Enter burst time:");
                scanf("%d",&b[i]);
                b1[i]=b[i];
        }

        int tq;

        printf("Enter Time
        Quantum:");
        scanf("%d",&tq);
        for(i=0;i<n-1;i++){

                for(j=i+1;j<n;j++){
```

```
                    if(a[i]>a[j]){

                        temp=a[i]
                        ; a[i]=a[j];
                        a[j]=temp
                        ;
                        temp=p[i]
                        ; p[i]=p[j];
                        p[j]=temp
                        ;
                        temp=b[i]
                        ; b[i]=b[j];
                        b[j]=temp
                        ;
                }
  }

}

        q[++r]=p[0];

        visited[0]=1

        ; int

        cq=a[0];

        while(f!=r){

                int
                x=q[++f];
                int y=b[x-
                1];
                b[x-1]-=tq;

                if(b[x-1]>=0){

                   cq+=tq;
                   ct[x-
                   1]=cq;
                }

                else{

                        cq+=y;
```

```
                ct[x-1]=cq;
            }
            for(i=0;i<n;i++){
                if(visited[i]==0 &&
                  a[i]<=cq){ q[++r]=p[i];
                  visited[i]=1;
                 }
            }
            if(b[x-1]>0)
              q[++r]=p[x-1];
printf("%d-->",x);
    }
    float
    w=0.0,t=0.0;
    for(i=0;i<n;i++){
            tat[i]=ct[i]-a[i];
            wt[i]=tat[i]-b1[i];
            w+=wt[i];
            t+=tat[i];
    }
    w/=n;
    t/=n;
    printf("\nPID\tAT\tBT\tCT\tTAT\tWT\n"
    ); for(i=0;i<n;i++)
    printf("%d\t%d\t%d\t%d\t%d\t%d\n",p[i],a[i],b1[i],ct[i],tat[i],wt[i])
    ; printf("Average Waiting Time=%.2f\n",w);
    printf("Average Turn Around Time=%.2f",t);
```

```
    return 0;
}
```

INPUT:

Enter no of process:

5 Enter process ID:1

Enter arrival time:0

Enter burst time:5

Enter process ID:2

Enter arrival time:1

Enter burst time:3

Enter process ID:3

Enter arrival time:2

Enter burst time:1

Enter process ID:4

Enter arrival time:3

Enter burst time:2

Enter process ID:5

Enter arrival time:4

Enter burst time:3

Enter Time

Quantum:2

OUTPUT:

1-->2-->3-->1-->4-->5-->2-->1-->5-->

PID	AT	BT	CT	TAT	WT
1	0	5	13	13	8
2	1	3	12	11	8
3	2	1	5	3	2
4	3	2	9	6	4
5	4	3	14	10	7

Average Waiting Time=5.80

Average Turn Around

Time=8.60

07.Priority Non Preemptive Algorithm

```
#include<stdio.h
> int main(){
        int n,temp;
        printf("Enter no of process:
        "); scanf("%d",&n);
        int
        p[n],a[n],b[n],q[n],ct[n],tat[n],wt[n],r[n],x[n];
        int i,j;
        for(i=0;i<n;i++){
                printf("Enter process
                ID:"); scanf("%d",&p[i]);
                printf("Enter arrival
                time:"); scanf("%d",&a[i]);
                printf("Enter burst time:");
                scanf("%d",&b[i]);
                printf("Enter priority:");
                scanf("%d",&q[i]);
                r[i]=999;
                x[i]=999;
        }
        int start=0;
        int order[n];
        int t=0;
        while(1){
            int f=0;
            for(i=0;i<n;i++
            ){
```

```
      if(r[i]!=0)
          f=1;
      }
   if(f==0) break;
   int min=999;
   int
   num,flag=0;
   for(j=0;j<n;j++
   )
            if(r[j]==999 && a[j]<=t){
               r[j]=b[j];
               x[j]=q[j];
             }
 for(j=0;j<n;j+
+) if(r[j]!=999)
     flag=1;
if(flag==0
   ){ t++;
    continue;
}
   for(j=0;j<n;j++){
            if(x[j]<min)
              {
              min=x[j];
              num=j;
             }
   }
   t+=r[num]
   ;
   r[num]=0;
```

```
        x[num]=999;
        ct[num]=t;
        order[start++]=nu
        m;
    }
        int sum=0;
        float
        tt=0.0,w=0.0;
        for(i=0;i<n;i++){
                tat[i]=ct[i]-a[i];
                wt[i]=tat[i]-b[i];
                tt+=tat[i];
                w+=wt[i];
        }
        tt=tt/n;
        w=w/n;
        printf("PID\tAT\tBT\tPT\tCT\tTAT\tWT\n
        "); for(i=0;i<n;i++){
             int k=order[i];
            printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n",p[k],a[k],b[k],q[k],ct[k],tat[k],wt[k]
            );
        }
       printf("Average Waiting Time=%.2f\n",w);
        printf("Average Turn Around
        Time=%.2f",tt); return 0;
}
```

INPUT:

```
Enter no of process:
3 Enter process
ID:1
```

Enter arrival

time:0 Enter burst

time:10 Enter

priority:0 Enter

process ID:2

Enter arrival

time:0 Enter burst

time:5 Enter

priority:2 Enter

process ID:3

Enter arrival

time:0 Enter burst

time:8 Enter

priority:1

OUTPUT:

PID	AT	BT	PT	CT	TAT	WT
1	0	10	0	10	10	0
3	0	8	1	18	18	10
2	0	5	2	23	23	18

Average Waiting Time=9.33

Average Turn Around

Time=17.00

8. Priority Preemptive Algorithm

```
#include<stdio.h
> int main(){
        int n,temp;
        printf("Enter no of process:
        "); scanf("%d",&n);
        int
        p[n],a[n],b[n],q[n],ct[n],tat[n],wt[n],r[n],x[n];
        int i,j;
        for(i=0;i<n;i++){
                printf("Enter process
                ID:"); scanf("%d",&p[i]);
                printf("Enter arrival
                time:"); scanf("%d",&a[i]);
                printf("Enter burst time:");
                scanf("%d",&b[i]);
                printf("Enter priority:");
                scanf("%d",&q[i]);
                r[i]=999;
                x[i]=999;
        }
        int max=-
        1,m=999;
        for(i=0;i<n;i++){
           if(a[i]>max)
              max=a[i];
          if(a[i]<m)
             m=a[i];
```

```
        }
while(1){
        int f=0;
        for(i=0;i<n;i++
        ){ if(r[i]!=0)
            f=1;
         }
        if(f==0)
        break; int
        min=999; int
        num;
        for(j=0;j<n;j+
        +)
                if(r[j]==999 && a[j]<=t){
                   r[j]=b[j];
                   x[j]=q[j];
                 }
        for(j=0;j<n;j++){
                if(x[j]<min)
                  {
                  min=x[j];
                  num=j;
                 }
        }
        if(t<max){
                r[num]-=1;
           t++;
         }
        else{
```

```
            t+=r[num]
            ;
            r[num]=0;
        }
        if(r[num]==0)
            x[num]=999
            ;
        ct[num]=t;
        printf("%d-->",num+1);
    }
        int sum=0;
        float
        tt=0.0,w=0.0;
        for(i=0;i<n;i++){
                tat[i]=ct[i]-a[i];
                wt[i]=tat[i]-b[i];
                tt+=tat[i];
                w+=wt[i];
        }
        tt=tt/n;
        w=w/n;
        printf("\nPID\tAT\tBT\tPT\tCT\tTAT\tWT\n"
        ); for(i=0;i<n;i++)
        printf("%d
        \t%d\t%d\t%d\t%d\t%d\t%d\n",p[i],a[i],b[i],q[i],ct[i],tat[i],wt[i]);
        printf("Average Waiting Time=%.2f\n",w);
        printf("Average Turn Around
        Time=%.2f",tt); return 0;
}
```

INPUT:

Enter priority:2

Enter process ID:2

Enter arrival

time:5 Enter burst

time:28 Enter

priority:0 Enter

process ID:3 Enter

arrival time:12

Enter burst time:2

Enter priority:3

Enter process ID:4

Enter arrival

time:2 Enter burst

time:10 Enter

priority:1 Enter

process ID:5 Enter

arrival time:9

Enter burst

time:16 Enter

priority:4

OUTPUT:

1-->1-->4-->4-->4-->2-->2-->2-->2-->2-->2-->2-->2-->4-->1-->3-->4-->

PID	AT	BT	PT	CT	TAT	WT
1	0	11	2	49	49	38
2	5	28	0	33	28	0
3	12	2	3	51	39	37
4	2	10	1	40	38	28
5	9	16	4	67	58	42

Average Waiting Time=29.00

Average Turn Around

Time=42.40

9. Worst Fit Technique

```
#include

<stdio.h> int

main(){

        int bn,pn,i,j;

        printf("Enter no of
        blocks:");
        scanf("%d",&bn);
        int blo[bn];

        printf("Enter the size of each
        block:"); for(i=0;i<bn;i++)
          scanf("%d",&blo[i]);

        printf("Enter no of

        processes:");

        scanf("%d",&pn);

        int pro[bn],alloc[bn];

        printf("Enter the size of each

        process:"); for(i=0;i<pn;i++){

          scanf("%d",&pro[i]

          ); alloc[i]=-1;

         }

        int windex=-1;
        for(i=0;i<pn;i++
        ){
                windex=-1;
                for(j=0;j<bn;j++
                ){
                        if(blo[j]>=pro[i]){

                                if(windex==-

                                  1)

                                  windex=j;

                                else
```

```
                        if(blo[windex]<blo[j]
                          ) windex=j;
                    }
                }
                blo[windex]-
              =pro[i]; alloc[i]=windex;
          }
          printf("Process Size\tBlock
          No\n"); for(i=0;i<pn;i++){
                if(alloc[i]!=-1)
                  printf("\t%d\t\t%d\n",pro[i],alloc[i]+1);
                else
                  printf("\t%d\t\t\tNot Allocated\n",pro[i]);
          }
          return 0;
}
```

OUTPUT:

```
Enter no of blocks:5
Enter the size of each block:100 500 200
300 600 Enter no of processes:4
Enter the size of each process:212 417
112 426 Process Size      Block No
     212          5
     417          2
     112          5
     426          Not Allocated
```

10. Best Fit Technique

```
#include

<stdio.h> int

main(){

        int bn,pn,i,j;

        printf("Enter no of

        blocks:");

        scanf("%d",&bn);

        int blo[bn];

        printf("Enter the size of each
        block:"); for(i=0;i<bn;i++)
          scanf("%d",&blo[i]);
        printf("Enter no of
        processes:");
        scanf("%d",&pn);
        int pro[bn],alloc[bn];

        printf("Enter the size of each
        process:"); for(i=0;i<pn;i++){
          scanf("%d",&pro[i]

          ); alloc[i]=-1;

         }

        int bindex=-1;

        for(i=0;i<pn;i++

        ){

                bindex=-1;

                for(j=0;j<bn;j++

                ){

                        if(blo[j]>=pro[i]){

                                if(bindex==-
                                  1)
                                  bindex=j;
                                else
```

```
                    if(blo[bindex]>blo[j]
                      ) bindex=j;
                }
            }
            blo[bindex]-
          =pro[i]; alloc[i]=bindex;
        }
        printf("Process Size\tBlock
        No\n"); for(i=0;i<pn;i++){
            if(alloc[i]!=-1)
              printf("\t%d\t\t\t%d\n",pro[i],alloc[i]+1);
            else
              printf("\t%d\t\t\tNot Allocated\n",pro[i]);
        }
        return 0;
}
```

OUTPUT:

Enter no of blocks:5

Enter the size of each block:100 500 200

300 600 Enter no of processes:4

Enter the size of each process:212 417 112 426

Process Size	Block No
212	4
417	2
112	3
426	5

11. First Fit Technique

```
#include

<stdio.h> int

main(){

        int bn,pn,i,j;

        printf("Enter no of
        blocks:");
        scanf("%d",&bn);
        int blo[bn];

        printf("Enter the size of each
        block:"); for(i=0;i<bn;i++)
          scanf("%d",&blo[i]);
        printf("Enter no of
        processes:");
        scanf("%d",&pn);
        int pro[bn],alloc[bn];

        printf("Enter the size of each

        process:"); for(i=0;i<pn;i++){

          scanf("%d",&pro[i]

          ); alloc[i]=-1;

         }

        for(i=0;i<pn;i++){

                for(j=0;j<bn;j++){

                        if(blo[j]>=pro[i]){

                                blo[j]-=pro[i];

                                alloc[i]=j;

                                break;

                        }

                }
```

```
        }
        printf("Process Size\tBlock
        No\n"); for(i=0;i<pn;i++){
                if(alloc[i]!=-1)
                  printf("\t%d\t\t\t%d\n",pro[i],alloc[i]+1);
                else
                  printf("\t%d\t\t\tNot Allocated\n",pro[i]);
        }
        return 0;
}
```

INPUT:

Enter no of blocks:5

Enter the size of each block:100 500 200

300 600 Enter no of processes:4

Enter the size of each process:212 417 112 426

OUTPUT:

Process Size	Block No
212	2
417	5
112	2
426	Not Allocated

12. MVT Technique

```
#include<stdio.h

> int main()

{

int ms,mp[10],i,

temp,n=0; char ch = 'y';

printf("\nEnter the total memory available (in Bytes)-

- "); scanf("%d",&ms);

temp=ms;

for(i=0;ch=='y';i++,n+

+)

{

printf("\nEnter memory required for process %d (in Bytes) --

",i+1); scanf("%d",&mp[i]);

if(mp[i]<=temp){

printf("\nMemory is allocated for Process %d

",i+1); temp = temp - mp[i];

}

else{

printf("\nMemory is
Full"); break;
}

printf("\nDo you want to continue(y/n) --
"); scanf(" %c", &ch);
}

printf("\nTotal Memory Available -- %d", ms);

printf("\n\tPROCESS\t\t MEMORY

ALLOCATED ");
```

```
for(i=0;i<n;i++)
    printf("\n \t%d\t\t%d",i+1,mp[i]);
printf("\nTotal Memory Allocated is %d",ms-
temp);
printf("\nTotal External Fragmentation is
%d",temp); return 0;
}
```

OUTPUT:

```
Enter the total memory available (in Bytes)--
1000 Enter memory required for process 1 (in
Bytes) -- 400 Memory is allocated for Process 1
Do you want to continue(y/n) -- y
Enter memory required for process 2 (in Bytes) --
275 Memory is allocated for Process 2
Do you want to continue(y/n) -- y
Enter memory required for process 3 (in Bytes) --
550 Memory is Full
Total Memory Available -- 1000
```

PROCESS	MEMORY ALLOCATED
1	400
2	275

```
Total Memory Allocated is 675
Total External Fragmentation is
325
```

13. MFT Technique

```
#include<stdio.h

> int main()

{

  int ms, bs, nob, ef,n,

  mp[10],tif=0; int i,p=0;

  printf("Enter the total memory available (in Bytes) -

  - "); scanf("%d",&ms);

  printf("Enter the block size (in Bytes) -

  - "); scanf("%d", &bs);

  nob=(ms/bs);

  ef=ms -

  nob*bs;

  printf("\nEnter the number of processes --
  "); scanf("%d",&n);
  for(i=0;i<n;i++)

  {

    printf("Enter memory required for process %d (in Bytes)--
    ",i+1); scanf("%d",&mp[i]);
  }

  printf("\nNo. of Blocks available in memory -- %d",nob);

  printf("\n\nPROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL
  FRAGMENTATION");

  for(i=0;i<n && p<nob;i++)

  {

    printf("\n
    %d\t\t%d",i+1,mp[i]); if(mp[i]
    > bs)
      printf("\t\tNO\t\t---");
```

```
    else

    {

      printf("\t\tYES\t%d",bs-
      mp[i]); tif = tif + bs-mp[i];
      p++;

    }

  }

  if(i<n)

    printf("\nMemory is Full, Remaining Processes cannot be
  accomodated"); printf("\n\nTotal Internal Fragmentation is %d",tif);
  printf("\nTotal External Fragmentation is
  %d",ef); return 0;
}
```

OUTPUT:

Enter the total memory available (in Bytes) -- 1000 Enter the block size (in Bytes) -- 300
Enter the number of processes -- 5

Enter memory required for process 1 (in Bytes)-- 275 Enter memory required for process 2 (in Bytes)-- 400 Enter memory required for process 3 (in Bytes)-- 290 Enter memory required for process 4 (in Bytes)-- 293 Enter memory required for process 5 (in Bytes)-- 100 No. of Blocks available in memory -- 3

PROCESS	MEMORY REQUIRED	ALLOCATED	INTERNAL FRAGMENTATION
1	275	YES	25
2	400	NO	---
3	290	YES	10
4	293	YES	7

Memory is Full, Remaining Processes cannot be

accomodated. Total Internal Fragmentation is 42

Total External Fragmentation is 100

14. Bankers Safety Sequence Algorithm

```
#include

<stdio.h> int

main(){

        int n,m,i,j;

        printf("Enter no of

        process:"); scanf("%d",&n);

        printf("Enter no of

        resources:");

        scanf("%d",&m);

        int

        alloc[n][m],max[n][m],need[n][m];

        int finish[n],avail[m],total[m];

        /*for(i=0;i<n;i++){

                printf("PROCESS

                %d\n",i+1); finish[i]=0;

                for(j=0;j<m;j++){

   printf("Max Value for Resource
   %d:",j+1); scanf("%d",&max[i][j]);
   printf("Allocated Value for Resource
   %d:",j+1); scanf("%d",&alloc[i][j]);
   need[i][j]=max[i][j]-alloc[i][j];

   }

  }*/

  printf("Enter Max
  Matrix:\n"); for(i=0;i<n;i++)
     for(j=0;j<m;j++)
       scanf("%d",&max[i][j]
       );
  printf("Enter Allocation Matrix:\n");
```

```
for(i=0;i<n;i++
      ){
      finish[i]=0;
   for(j=0;j<m;j++){
     scanf("%d",&alloc[i][j]
     );
     need[i][j]=max[i][j]-alloc[i][j];
   }
}
for(j=0;j<m;j++){
          avail[j]=0;
                for(i=0;i<n;i++){
                        avail[j]+=alloc[i][j];
                }
       }
for(i=0;i<m;i++){
         printf("Total value of Resource
         %d:",i+1); scanf("%d",&total[i]);
         avail[i]=total[i]-avail[i];
       }
       int
x,order[n],count=0;
for(x=0;x<n;x++){
       for(i=0;i<n;i++){
                if(finish[i]==0){
                        int flag=0;
                        for(j=0;j<m;j++
                        ){
                                if(need[i][j]>avail[j])
                             { flag=1;
```

```
                    break;
                        }
                }
                if(flag==0){
                        order[count++]=i+
                        1; finish[i]=1;
                        for(j=0;j<m;j++)
                          avail[j]+=alloc[i][j];
                }
        }
}
   }
      count=0;
   for(i=0;i<n;i+
   +)
            if(finish[i]==1
              ) count++;
   if(count==n){
     printf("Safe
     State\n");
     printf("Order of
     Execution:\n");
     for(i=0;i<n;i++)
   printf("Process %d-->",order[i]);
   }
      else
        printf("Unsafe
   State"); return 0;
```

}

INPUT:

Enter no of process:5

Enter no of

resources:3 Enter

Max Matrix:

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

Enter Allocation
Matrix:

0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

Total value of Resource

1:10 Total value of

Resource 2:5 Total value

of Resource 3:7

OUTPUT:

Safe State

Order of Execution:

Process 2-->Process 4-->Process 5-->Process 1-->Process 3-->

15. Bankers Resource Request Algorithm

```
#include

<stdio.h> int

main(){

        int n,m,i,j;

        printf("Enter no of

        process:"); scanf("%d",&n);

        printf("Enter no of

        resources:");

        scanf("%d",&m);

        int

        alloc[n][m],max[n][m],need[n][m];

        int finish[n],avail[m],total[m];

        printf("Enter Max Matrix:\n");

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

                finish[i]=0;
                for(j=0;j<m;j++
                ){
     scanf("%d",&max[i][j]);

    }

}

printf("Enter Allocated
      Matrix:\n"); for(i=0;i<n;i++){
                for(j=0;j<m;j++
    ){
    scanf("%d",&alloc[i][j]);
    need[i][j]=max[i][j]-alloc[i][j];

    }

}

for(j=0;j<m;j++){

          avail[j]=0;
```

```
            for(i=0;i<n;i++)
                    avail[j]+=alloc[i][j];
        }
for(i=0;i<m;i++){
        printf("Total value of Resource
        %d:",i+1); scanf("%d",&total[i]);
        avail[i]=total[i]-avail[i];
        }
        //Request
        Algorithm int
        r[m],p;
        printf("Enter Process
        No:"); scanf("%d",&p);
        printf("Enter Request
        Vector:"); for(i=0;i<m;i++)
          scanf("%d",&r[i]
        );
        for(i=0;i<m;i++){
            if(r[i]<=need[p-1][i]
                &&r[i]<=avail[i]){ avail[i]-
                =r[i];
                alloc[p-1][i]+=r[i];
                need[p-1][i]-=r[i];
           }
           else{
       printf("Process must
       wait"); return 0;
           }
        }
```

```
 int x,flag,f;
for(x=0;x<n;x++
){
    for(i=0;i<n;i++){
        if(finish[i]==0)
        {
            flag=0;
            for(j=0;j<m;j++
            ){
                if(need[i][j]>avail[j])
            { flag=1;
            break;
                }
            }
            if(flag==0){
                printf("Process
                %d\n",i+1); finish[i]=1;
                for(j=0;j<m;j++)
                avail[j]+=alloc[i][j];
                break;
            }
    }
  }
   }
f=0;
 for(i=0;i<n;i++)
   if(finish[i]==0
   )
    f=1;
     if(f==0)
```

```
            printf("Resources Allocated
             Successfully"); else
               printf("Process must
          wait"); return 0;
}
```

INPUT:

Enter no of process:5

Enter no of

resources:3 Enter

Max Matrix:

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

Enter Allocated
Matrix:

0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

Total value of Resource

1:10 Total value of

Resource 2:5 Total value

of Resource 3:7 Enter

Process No:2

Enter Request Vector:1 0 2

OUTPUT:

Process 2

Process 4

Process 1

Process 3

Process 5

Resources Allocated Successfully

16. Optimal Page Replacement Algorithm

```
#include

<stdio.h> int

main(){

        int m,n,k,i,index,l,max;
        printf("Enter number of
        frames:"); scanf("%d",&m);
        printf("Enter length of page
        string:"); scanf("%d",&n);
        int p[n];

        printf("Enter page reference
        string:"); for(i=0;i<n;i++)
          scanf("%d",&p[i]);

        int pf=0,j,flag1=0,flag2=0;

        int

        fr[m],fs[m],lg[m],found,h=0;

        for(i=0;i<m;i++)

          fr[i]=-1;

  for(j=0;j<n;j+

  +)

  {

   flag1=0,flag2=
   0;
   for(i=0;i<m;i++
   )
   {

     if(fr[i]==p[j])

     {

      flag1=

      1;

      flag2=

      1; h++;
```

```
      break;
    }
 }
if(flag1==0)
{
   for(i=0;i<m;i++)
  {
    if(fr[i]==-1)
   {
   fr[i]=p[j];
   flag2=1;
   pf++;
   break;
  }
  }
}
if(flag2==0)
 {
  for(i=0;i<m;i+
     +) lg[i]=0;
  for(i=0;i<m;i++)
  {
    for(k=j+1;k<=n;k++)
     {
     if(fr[i]==p[k])
      {
```

```
    lg[i]=k-
    j;
    break;
  }
 }
}
found=0;
for(i=0;i<m;i+
+)
{
if(lg[i]==0)
{
 index=i;
 found =
 1; break;
 }
}
if(found==0)
{
max=lg[0];
index=0;
for(i=0;i<m;i+
+)
{
if(max<lg[i])
{
max=lg[i]
; index=i;
}
```

```
      }
     }
     fr[index]=p[j]
     ; pf++;
     }
    for(i=0;i<m;i++)
      printf("%d\t",fr[i]);
    printf("\n");
  }
        printf("Number of page
        faults:%d\n",pf); printf("Number of
        hits:%d\n",h);
        return 0;
}
```

INPUT:

Enter number of frames:3

Enter length of page

string:12

Enter page reference string:1 2 3 4 1 2 5 1 2 3 4 5

OUTPUT:

1 -1 -1

1 2 -1

1 2 3

1 2 4

1 2 4

1 2 4

1 2 5

1 2 5

1 2 5

3 2 5

4 2 5

4 2 5

Number of page

faults:7 Number of

hits:5

17. LRU Page Replacement Algorithm

```
#include
<stdio.h> int
main(){
        int m,n,k,i,index,l,max;
        printf("Enter number of
        frames:"); scanf("%d",&m);
        printf("Enter length of page
        string:"); scanf("%d",&n);
        int p[n];
        printf("Enter page reference
        string:"); for(i=0;i<n;i++)
          scanf("%d",&p[i]);
        int pf=0,j,flag1=0,flag2=0;
        int
        fr[m],fs[m],lg[m],found,h=0;
        for(i=0;i<m;i++)
          fr[i]=-1;
  for(j=0;j<n;j+
  +)
  {
    flag1=0,flag2=
    0;
    for(i=0;i<m;i++
    )
    {
      if(fr[i]==p[j])
      {
        flag1=
        1;
        flag2=
        1; h++;
```

```
      break;
    }
 }
if(flag1==0)
{
   for(i=0;i<m;i++)
  {
    if(fr[i]==-1)
   {
   fr[i]=p[j];
   flag2=1;
   pf++;
   break;
  }
  }
}
if(flag2==0)
 {
  for(i=0;i<m;i+
     +) lg[i]=0;
  for(i=0;i<m;i++)
  {
    for(k=j-1;k>=0;k--)
     {
     if(fr[i]==p[k])
      {
```

```
    lg[i]=j-
    k;
    break;
  }
 }
}
found=0;
for(i=0;i<m;i+
+)
{
if(lg[i]==0)
{
 index=i;
 found =
 1; break;
 }
}
if(found==0)
{
max=lg[0];
index=0;
for(i=0;i<m;i+
+)
{
if(max<lg[i])
{
max=lg[i]
; index=i;
}
```

```
        }
       }
       fr[index]=p[j]
       ; pf++;
       }
      for(i=0;i<m;i++)
        printf("%d\t",fr[i]);
      printf("\n");
    }
          printf("Number of page
          faults:%d\n",pf); printf("Number of
          hits:%d\n",h);
          return 0;
}
```

INPUT:

Enter number of frames:4

Enter length of page

string:13

Enter page reference string:7 0 1 2 0 3 0 4 2 3 0 3 2

OUTPUT:

7 -1 -1 -1

7 0 -1 -1

7 0 1 -1

7 0 1 2

7 0 1 2

3 0 1 2

3 0 1 2

3 0 4 2

3 0 4 2

3 0 4 2

3 0 4 2

3 0 4 2

3 0 4 2

Number of page

faults:6 Number of

hits:7

18. FIFO Page Replacement Algorithm

```
#include
<stdio.h> int
main(){
        int n,k,i;
        printf("Enter number of
        frames:"); scanf("%d",&k);
        printf("Enter length of page
        string:"); scanf("%d",&n);
        int page[n];
        printf("Enter page reference
        string:"); for(i=0;i<n;i++)
          scanf("%d",&page[i])
        ; int
        pf=0,j,f1=0,f2=0,h=0;
        int fr[k];
        int l=-1;
        for(i=0;i<k;i+
        +)
          fr[i]=-1;
        for(i=0;i<n;i++
        ){
                f1=0
                ;
                f2=0
                ;
                for(j=0;j<k;j++)
                  if(page[i]==fr[j])
                  {
                    f1=1
                    ;
                    h++;
                    break;
                  }
```

```
            if(f1==0){
                pf++;
                for(j=0;j<k;j++
                ){
                    if(fr[j]==-1){
                      fr[j]=page[i]
                      ; f2=1;
                      break;
                     }
                }
                if(f2==0)
                {
                    l=(l+1)%k;
                    fr[l]=page[i]
                    ;
                }
            }
            for(j=0;j<k;j++)
              printf("%d\t",fr[j]);
            printf("\n");
        }
        printf("Number of page
        faults:%d\n",pf); printf("Number of
        hits:%d",h);
        return 0;
}
```

INPUT:

Enter number of frames:3

Enter length of page

string:7

Enter page reference string:1 3 0 3 5 6 3

OUTPUT:

1 -1 -1

1 3 -1

1 3 0

1 3 0

5 3 0

5 6 0

5 6 3

Number of page

faults:6 Number of

hits:1

19. FCFS(First Come First Served) disk scheduling algorithm

```
#include<stdio.h
> #include
<stdlib.h> int
main()
{
    int
    n,head,i,j,k,seek=0,max,diff;
    float avg;
    printf("Enter the max range of
    disk:"); scanf("%d",&max);
    printf("Enter the size of queue
    request:"); scanf("%d",&n);
    int queue[n+1];

    printf("Enter the queue of disk positions to be

    read:"); for(i=1;i<=n;i++)

      scanf("%d",&queue[i]);

    printf("Enter the initial head

    position:"); scanf("%d",&head);

    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 time %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
       %.2f\n",avg); return 0;
}
```

INPUT:

```
Enter the max range of
disk:200 Enter the size of
queue request:8
Enter the queue of disk positions to be read:176 79 34 60 92 11
41 114 Enter the initial head position:50
```

OUTPUT:

Disk head moves from 50 to 176 with seek time 126 Disk head moves from 176 to 79 with seek time 97 Disk head moves from 79 to 34 with seek time 45 Disk head moves from 34 to 60 with seek time 26 Disk head moves from 60 to 92 with seek time 32 Disk head moves from 92 to 11 with seek time 81 Disk head moves from 11 to 41 with seek time 30 Disk head moves from 41 to 114 with seek time 73 Total seek time is 510

Average seek time is 63.75

20. SCAN Disk Scheduling Algorithm

```
#include<stdio.h
> #include
<stdlib.h> int
main()
{
    int
    n,head,i,j,k,seek=0,max,diff;
    float avg;
    printf("Enter the max range of
    disk:"); scanf("%d",&max);
    printf("Enter the size of queue
    request:"); scanf("%d",&n);
    int queue[n+2];
    printf("Enter the queue of disk positions to be
    read:"); for(i=1;i<=n;i++)
      scanf("%d",&queue[i]);
    printf("Enter the initial head
    position:"); scanf("%d",&head);
    queue[0]=head;
    queue[n+1]=max-
    1; int temp,pos;
    for(i=0;i<n+2;i++){
          for(j=i+1;j<n+2;j++){
                if(queue[i]>queue[j]){
                      temp=queue[i];
                      queue[i]=queue[j
                      ]; queue[j]=temp;
```

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

INPUT:

Enter the max range of

disk:200 Enter the size of

queue request:8

Enter the queue of disk positions to be read:90 120 35 122 38 128

65 68 Enter the initial head position:50

OUTPUT:

Disk head moves from 50 to 65 with seek time

15 Disk head moves from 65 to 68 with seek

time 3 Disk head moves from 68 to 90 with

seek time 22 Disk head moves from 90 to 120

with seek time 30 Disk head moves from 120 to

122 with seek time 2 Disk head moves from

122 to 128 with seek time 6 Disk head moves

from 128 to 199 with seek time 71 Disk head

moves from 199 to 38 with seek time 161 Disk

head moves from 38 to 35 with seek time 3

Total seek time is 313

Average seek time is 39.13

21.C-SCAN Disk Scheduling Algorithm

```
#include<stdio.h
> #include
<stdlib.h> int
main()
{
    int
    n,head,i,j,k,seek=0,max,diff;
    float avg;
    printf("Enter the max range of
    disk:"); scanf("%d",&max);
    printf("Enter the size of queue
    request:"); scanf("%d",&n);
    int queue[n+2];
    printf("Enter the queue of disk positions to be
    read:"); for(i=1;i<=n;i++)
      scanf("%d",&queue[i]);
    printf("Enter the initial head
    position:"); scanf("%d",&head);
    queue[0]=head;
    queue[n+1]=max-
    1; int temp,pos;
    for(i=0;i<n+2;i++){
          for(j=i+1;j<n+2;j++){
                if(queue[i]>queue[j]){
                      temp=queue[i];
                      queue[i]=queue[j
                      ]; queue[j]=temp;
```

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

INPUT:

Enter the max range of

disk:200 Enter the size of

queue request:8

Enter the queue of disk positions to be read:90 120 35 122 38 128

65 68 Enter the initial head position:50

OUTPUT:

Disk head moves from 50 to 65 with seek time

15 Disk head moves from 65 to 68 with seek

time 3 Disk head moves from 68 to 90 with

seek time 22 Disk head moves from 90 to 120

with seek time 30 Disk head moves from 120 to

122 with seek time 2 Disk head moves from

122 to 128 with seek time 6 Disk head moves

from 128 to 199 with seek time 71 Disk head

moves from 0 to 35 with seek time 35 Disk

head moves from 35 to 38 with seek time 3

Total seek time is 386

Average seek time is 48.25

22.Producer Consumer Problem

```
#include <pthread.h>
#include
<semaphore.h>
#include <stdlib.h>
#include <stdio.h>
#define MaxItems 5
#define
BufferSize 5
sem_t empty;
sem_t full;
int in = 0;

int out =

0;

int buffer[BufferSize];

pthread_mutex_t mutex;

void *producer(void

*pno)

{

   int item;

   for(int i = 0; i < MaxItems; i++)

      { item = rand();

      sem_wait(&empty);

      pthread_mutex_lock(&mute

      x); buffer[in] = item;

      printf("Producer %d: Insert Item %d at %d\n", *((int

      *)pno),buffer[in],in); in = (in+1)%BufferSize;

      pthread_mutex_unlock(&mute

      x); sem_post(&full);

   }
```

```
}

void *consumer(void *cno)
{

   for(int i = 0; i < MaxItems; i++)
      { sem_wait(&full);
      pthread_mutex_lock(&mute
      x); int item = buffer[out];
      printf("Consumer %d: Remove Item %d from %d\n",*((int *)cno),item,
      out); out = (out+1)%BufferSize;
      pthread_mutex_unlock(&mute

      x); sem_post(&empty);

   }
}
int main()

{

   pthread_t pro[5],con[5];
   pthread_mutex_init(&mutex,
   NULL);
   sem_init(&empty,0,BufferSize);
   sem_init(&full,0,0);
   int a[5] = {1,2,3,4,5};

   for(int i = 0; i < 5; i++) {

      pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);

   }
   for(int i = 0; i < 5; i++) {

      pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
   }

   for(int i = 0; i < 5; i++) {
      pthread_join(pro[i],
      NULL);
   }

   for(int i = 0; i < 5; i++) {
      pthread_join(con[i], NULL);
   }
   pthread_mutex_destroy(&mute
   x); sem_destroy(&empty);
   sem_destroy(&full);
   return 0;
}
```

OUTPUT:

Producer 1: Insert Item 41 at 0

Producer 1: Insert Item 18467 at 1

Producer 1: Insert Item 6334 at 2

Producer 2: Insert Item 41 at 3

Producer 1: Insert Item 26500 at 4

Consumer 1: Remove Item 41 from 0

Consumer 1: Remove Item 18467 from 1

Producer 2: Insert Item 18467 at 0

Consumer 1: Remove Item 6334 from 2

Producer 1: Insert Item 19169 at 1

Consumer 1: Remove Item 41 from 3

Producer 3: Insert Item 41 at 2

Consumer 1: Remove Item 26500 from 4

Producer 4: Insert Item 41 at 3

Producer 5: Insert Item 41 at 4

Consumer 2: Remove Item 18467 from 0

Consumer 2: Remove Item 19169 from 1

Producer 2: Insert Item 6334 at 0

Consumer 2: Remove Item 41 from 2

Producer 3: Insert Item 18467 at 1

Consumer 2: Remove Item 41 from 3

Producer 4: Insert Item 18467 at 2

Consumer 2: Remove Item 41 from 4

Producer 5: Insert Item 18467 at 3

Consumer 3: Remove Item 6334 from 0

Producer 2: Insert Item 26500 at 4

Consumer 3: Remove Item 18467 from 1

Producer 3: Insert Item 6334 at 0

Consumer 3: Remove Item 18467 from 2

Producer 4: Insert Item 6334 at 1

Consumer 4: Remove Item 18467 from 3

Consumer 3: Remove Item 26500 from 4

Producer 5: Insert Item 6334 at 2

Consumer 4: Remove Item 6334 from 0

Producer 2: Insert Item 19169 at 3

Consumer 5: Remove Item 6334 from 1

Producer 3: Insert Item 26500 at 4

Consumer 3: Remove Item 6334 from 2

Producer 4: Insert Item 26500 at 0

Consumer 4: Remove Item 19169 from 3

Producer 5: Insert Item 26500 at 1

Consumer 5: Remove Item 26500 from 4

Producer 3: Insert Item 19169 at 2

Consumer 4: Remove Item 26500 from 0

Producer 4: Insert Item 19169 at 3

Consumer 5: Remove Item 26500 from 1

Producer 5: Insert Item 19169 at 4

Consumer 4: Remove Item 19169 from 2

Consumer 5: Remove Item 19169 from 3

Consumer 5: Remove Item 19169 from 4

24.Dining Philosopher Problem

```
#include<stdio.h>
#include<unistd.h>
#include<semaphore.
h>
#include<pthread.h>
#define N 5
#define THINKING 0

#define HUNGRY 1

#define EATING 2

#define LEFT
(ph_num+4)%N #define
RIGHT (ph_num+1)%N
sem_t mutex;
sem_t S[N];

void * philospher(void

*num); void take_fork(int);

void

put_fork(int);

void test(int);

int state[N];

int

phil_num[N]={0,1,2,3,4};

int main()

{

   int i;

   pthread_t

   thread_id[N];

   sem_init(&mutex,0,1);

   for(i=0;i<N;i++)

      sem_init(&S[i],0,0);
```

```
    for(i=0;i<N;i++)
    {
        pthread_create(&thread_id[i],NULL,philospher,&phil_nu
        m[i]); printf("Philosopher %d is thinking\n",i+1);
    }
    for(i=0;i<N;i++)
        pthread_join(thread_id[i],NUL
        L);
}
void *philospher(void *num)
{
    while(1)
    {
        int *i = num;
        sleep(1);
        take_fork(*i)
        ; sleep(0);
        put_fork(*i);
    }
}
void take_fork(int ph_num)
{
    sem_wait(&mutex);
    state[ph_num] =
    HUNGRY;
    printf("Philosopher %d is
    Hungry\n",ph_num+1); test(ph_num);
    sem_post(&mutex);
```

```
    sem_wait(&S[ph_num
    ]); sleep(1);
}
void test(int ph_num)
{
    if (state[ph_num] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
    {
        state[ph_num] =
        EATING; sleep(2);
        printf("Philosopher %d takes fork %d and
        %d\n",ph_num+1,LEFT+1,ph_num+1); printf("Philosopher %d is
        Eating\n",ph_num+1);
        sem_post(&S[ph_num]);
    }
}
void put_fork(int ph_num)
{
    sem_wait(&mutex);
    state[ph_num] =
    THINKING;
    printf("Philosopher %d putting fork %d and %d
    down\n",ph_num+1,LEFT+1,ph_num+1); printf("Philosopher %d is
    thinking\n",ph_num+1);
    test(LEFT);
    test(RIGHT);
    sem_post(&mutex
    );
}
```

OUTPUT:

Philosopher 1 is thinking

Philosopher 2 is thinking

Philosopher 3 is thinking

Philosopher 4 is thinking

Philosopher 5 is thinking

Philosopher 2 is Hungry

Philosopher 2 takes fork 1

and 2 Philosopher 2 is Eating

Philosopher 1 is Hungry

Philosopher 3 is Hungry

Philosopher 4 is Hungry

Philosopher 4 takes fork 3

and 4 Philosopher 4 is Eating

Philosopher 5 is Hungry

Philosopher 2 putting fork 1 and 2

down Philosopher 2 is thinking

Philosopher 1 takes fork 5

and 1 Philosopher 1 is Eating

Philosopher 4 putting fork 3 and 4

down Philosopher 4 is thinking

Philosopher 3 takes fork 2

and 3 Philosopher 3 is Eating

Philosopher 2 is Hungry

Philosopher 1 putting fork 5 and 1

down Philosopher 1 is thinking

Philosopher 5 takes fork 4

and 5 Philosopher 5 is Eating

Philosopher 4 is Hungry

Philosopher 3 putting fork 2 and 3

down Philosopher 3 is thinking

Philosopher 2 takes fork 1

and 2 Philosopher 2 is Eating

Philosopher 1 is Hungry

Philosopher 5 putting fork 4 and 5

down Philosopher 5 is thinking

^C

24. Readers Writers Problem

```
#include <pthread.h>
#include
<semaphore.h>
#include <stdio.h>
sem_t wrt;
pthread_mutex_t
mutex; int cnt = 1;
int numreader = 0;

void *writer(void *wno)

{

   sem_wait(&wrt)

   ; cnt = cnt*2;

   printf("Writer %d modified cnt to %d\n",(*((int

   *)wno)),cnt); sem_post(&wrt);



}

void *reader(void *rno)

{

   pthread_mutex_lock(&mute

   x); numreader++;

   if(numreader ==

      1) {

      sem_wait(&wrt);

   }

   pthread_mutex_unlock(&mutex);

   printf("Reader %d: read cnt as %d\n",*((int

   *)rno),cnt); pthread_mutex_lock(&mutex);
```

```
    numreader--;

    if(numreader ==

    0) {

        sem_post(&wrt);

    }

    pthread_mutex_unlock(&mutex);

}

int main()

{

    pthread_t read[10],write[5];
    pthread_mutex_init(&mutex,
    NULL); sem_init(&wrt,0,1);
    int a[10] = {1,2,3,4,5,6,7,8,9,10};

    for(int i = 0; i < 10; i++) {

        pthread_create(&read[i], NULL, (void *)reader, (void *)&a[i]);

    }

    for(int i = 0; i < 5; i++) {

        pthread_create(&write[i], NULL, (void *)writer, (void *)&a[i]);

    }

    for(int i = 0; i < 10; i++) {
        pthread_join(read[i],
        NULL);
    }

    for(int i = 0; i < 5; i++) {
        pthread_join(write[i],
        NULL);
    }
    pthread_mutex_destroy(&mutex);

     sem_destroy(&wrt); return 0;


}
```

OUTPUT:

Reader 2: read cnt as 1

Reader 1: read cnt as 1

Reader 5: read cnt as 1

Reader 8: read cnt as 1

Reader 4: read cnt as 1

Reader 6: read cnt as 1

Reader 9: read cnt as 1

Reader 10: read cnt as 1

Reader 3: read cnt as 1

Writer 1 modified cnt to 2

Writer 2 modified cnt to 4

Reader 7: read cnt as 4

Writer 3 modified cnt to 8

Writer 5 modified cnt to 16

Writer 4 modified cnt to 32

25. Sequential File Allocation Strategy

```
#include<stdio.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"); do{
    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);

    }while(c)

    ; return

    0;

}
```

OUTPUT:

Files Allocated are :

Enter starting block and length of files:

14 3 14 1

15 1

16 1

The file is allocated to disk

Do you want to enter more file(Yes - 1/No -

0)1 Enter starting block and length of files:

14 1 The file is not allocated

Do you want to enter more file(Yes - 1/No -

0)1 Enter starting block and length of files:

14 4 The file is not allocated

Do you want to enter more file(Yes - 1/No -

0)1 Enter starting block and length of files:

1 6

1 1

2 1

3 1

4 1

5 1

6 1

The file is allocated to disk

Do you want to enter more file(Yes - 1/No - 0)0

26. Indexed File Allocation Strategy

```
#include<stdio.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;

   do{

      printf("Enter the index block:
      "); scanf("%d",&ind);
      if(f[ind]!=1)

      {

       printf("Enter no of blocks and block nums needed for file with the index %d on the disk
:",ind);

       scanf("%d",&n

       ); count=0;

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

       {

          scanf("%d", &index[i]);

          if(f[index[i]]==0

          ) count++;

       }

       if(count==n){

           f[ind]=1;

           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("Do you want to enter more file(Yes - 1/No -
     0)"); scanf("%d", &c);
    }
    else{
       printf("%d index is already allocated \n",ind);
    }
     }while(c)
; return 0;
}
```

OUTPUT:

```
Enter the index block: 5

Enter no of blocks and block nums needed for file with the index 5 on the
disk :4 1 2 3 4
Allocated
File
Indexed 5
 -------->1 :
1
5-------->2 : 1

5-------->3 : 1


5-------->4 : 1

Do you want to enter more file(Yes - 1/No -

0)1 Enter the index block: 4
```

4 index is already

allocated Enter the index

block: 6

Enter no of blocks and block nums needed for file with the index 6 on the

disk :2 7 8

Allocated

File

Indexed 6

 -------->7 :

1

6-------->8 : 1

Do you want to enter more file(Yes - 1/No - 0)0

27. Linked File Allocation Strategy

```
#include<stdio.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"); do{
    count=0;
    printf("Enter starting block and length of files:
    "); scanf("%d %d", &st,&len);
    if(f[st]==0)
    { int fg=0;
    int
    arr[len],m=0;;
    for(k=st;;k++){
         if(f[k]==0){
         arr[m++]=k
          ; count++;
         }
         if(count==len)
           { fg=1;
           break;
         }
    }
    m=1
    ;
```

```
 if(fg==1)
 {
      printf("Block--->Nxt
   Block\n"); for(j=arr[0];j<arr[len-
   1];j++)
     if(f[j]==0)
     {
      f[j]=arr[m++];
      printf("%d--->%d\n",j,f[j]);
     }
   f[arr[len-1]]=999;
   printf("%d--->%d\n",arr[len-1],999);
   printf(" The file is allocated to
   disk\n");
 }
else
   printf("The file is not allocated \n");
}
else
   printf("Block is full\n");
printf("Do you want to enter more file(Yes - 1/No -
0)"); scanf("%d", &c);
}while(c)
; return
0;
}
```

OUTPUT:

Files Allocated are :

Enter starting block and length of files:

2 2 Block--->Nxt Block

2--->3

3--->999

The file is allocated to disk

Do you want to enter more file(Yes - 1/No -

0)1 Enter starting block and length of files:

3 4 Block is full

Do you want to enter more file(Yes - 1/No -

0)1 Enter starting block and length of files:

1 5 Block--->Nxt Block

1--->4

4--->5

5--->6

6--->7

7--->999

The file is allocated to disk

Do you want to enter more file(Yes - 1/No - 0)0