LS COMMAND

```c
#include<stdio.h>
#include<dirent.h>
main()
{
char dirname[10];
DIR*p;
struct dirent *d;
printf("Enter directory name\n");
scanf("%s",dirname);
p=opendir(dirname);
if(p==NULL)
  {
  perror("Cannot find directory");
  exit(-1);
  }
while(d=readdir(p))
  printf("%s\n",d->d_name);
}
```

CP COMMAND

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
int main(int argc, char **argv)
{
    char buffer[1024];
    int files[2];
    ssize_t count;
    /* Check for insufficient parameters */
    if (argc < 3)
        return -1;
    files[0] = open(argv[1], O_RDONLY);
    if (files[0] == -1) /* Check if file opened */
        return -1;
    files[1] = open(argv[2], O_WRONLY | O_CREAT | S_IRUSR | S_IWUSR);
    if (files[1] == -1) /* Check if file opened (permissions problems ...) */
    {
        close(files[0]);
        return -1;
    }
    while ((count = read(files[0], buffer, sizeof(buffer))) != 0)
        write(files[1], buffer, count);
    return 0;
}
```

FCFS

```c
#include <stdio.h>
int main()
{
    int pid[15];
    int bt[15];
    int n;
    printf("Enter the number of processes: ");
    scanf("%d",&n);

    printf("Enter process id of all the processes: ");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&pid[i]);
    }

    printf("Enter burst time of all the processes: ");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&bt[i]);
    }

    int i, wt[n];
    wt[0]=0;

    for(i=1; i<n; i++)
    {
        wt[i]= bt[i-1]+ wt[i-1];
    }

    printf("Process ID    Burst Time    Waiting Time    TurnAround Time\n");
    float twt=0.0;
    float tat= 0.0;
    for(i=0; i<n; i++)
    {
        printf("%d\t\t", pid[i]);
        printf("%d\t\t", bt[i]);
        printf("%d\t\t", wt[i]);
        printf("%d\t\t", bt[i]+wt[i]);
        printf("\n");
        twt += wt[i];
tat += (wt[i]+bt[i]);
    }
    float att,awt;
    awt = twt/n;
    att = tat/n;
    printf("Avg. waiting time= %f\n",awt);
    printf("Avg. turnaround time= %f",att);
}
```

SRT

```c
#include <stdio.h>
int main()
{
 int a[10],b[10],x[10],i,j,smallest,count=0,time,n;
 double avg=0,tt=0,end;
  printf("enter the number of Processes:\n");
  scanf("%d",&n);
 printf("enter arrival time\n");
 for(i=0;i<n;i++)
 scanf("%d",&a[i]);
 printf("enter burst time\n");
 for(i=0;i<n;i++)
 scanf("%d",&b[i]);
 for(i=0;i<n;i++)
 x[i]=b[i];

  b[9]=9999;

 for(time=0;count!=n;time++)
 {
  smallest=9;
 for(i=0;i<n;i++)
 {
  if(a[i]<=time && b[i]<b[smallest] && b[i]>0 )
  smallest=i;
 }
 b[smallest]--;
 if(b[smallest]==0)
 {
  count++;
  end=time+1;
  avg=avg+end-a[smallest]-x[smallest];
  tt= tt+end-a[smallest];
 }
 }
 printf("\n\nAverage waiting time = %lf\n",avg/n);
   printf("Average Turnaround time = %lf",tt/n);
   return 0;
}
```

Producer-consumer using semaphpores

```c
#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);
printf("\n remaining items in buffer is:%d%d"x+,x);
}
```

DINING PHILOSOPHER

```c
#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);
}
```

BANKERS ALGORITHM

```c
#include <stdio.h>
int main()
{
        int n, m, i, j, k;
        n = 5; // Number of processes
        m = 3; // Number of resources
        int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix
                                        { 2, 0, 0 }, // P1
                                        { 3, 0, 2 }, // P2
                                        { 2, 1, 1 }, // P3
                                        { 0, 0, 2 } }; // P4

        int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix
                                { 3, 2, 2 }, // P1
                                { 9, 0, 2 }, // P2
                                { 2, 2, 2 }, // P3
                                { 4, 3, 3 } }; // P4

        int avail[3] = { 3, 3, 2 };

        int f[n], ans[n], ind = 0;
        for (k = 0; k < n; k++) {
                f[k] = 0;
        }
        int need[n][m];
        for (i = 0; i < n; i++) {
                for (j = 0; j < m; j++)
                        need[i][j] = max[i][j] - alloc[i][j];
        }
        int y = 0;
        for (k = 0; k < 5; k++) {
                for (i = 0; i < n; i++) {
                        if (f[i] == 0) {

                                int flag = 0;
                                for (j = 0; j < m; j++) {
                                        if (need[i][j] > avail[j]){
                                                flag = 1;
                                                break;
                                        }
                                }

                                if (flag == 0) {
                                        ans[ind++] = i;
                                        for (y = 0; y < m; y++)
                                                avail[y] += alloc[i][y];
                                        f[i] = 1;
                                }
                        }
                }
```

```
    }

    int flag = 1;

    for(int i=0;i<n;i++)
    {
    if(f[i]==0)
    {
            flag=0;
            printf("The following system is not safe");
            break;
    }
    }

    if(flag==1)
    {
    printf("Following is the SAFE Sequence\n");
    for (i = 0; i < n - 1; i++)
            printf(" P%d ->", ans[i]);
    printf(" P%d", ans[n - 1]);
    }


    return (0);


}
```

LRU

```c
#include<stdio.h>
main()
{
        int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
        printf("Enter no of pages:");
        scanf("%d",&n);
        printf("Enter the reference string:");
        for(i=0;i<n;i++)
        scanf("%d",&p[i]);
        printf("Enter no of frames:");
        scanf("%d",&f);
        q[k]=p[k];
        printf("\n\t%d\n",q[k]);
        c++;
        k++;
        for(i=1;i<n;i++)
        {
                c1=0;
                for(j=0;j<f;j++)
                {
                        if(p[i]!=q[j])
                        c1++;
                }
                if(c1==f)
                {
                        c++;
                        if(k<f)
                        {
                                q[k]=p[i];
                                k++;
                                for(j=0;j<k;j++)
                                printf("\t%d",q[j]);
                                printf("\n");
                        }
                        else
                        {
                                for(r=0;r<f;r++)
                                {
                                        c2[r]=0;
                                        for(j=i-1;j<n;j--)
                                        {
                                        if(q[r]!=p[j])
                                        c2[r]++;
                                        else
                                        break;
                                        }
                                }
                        }
                        for(r=0;r<f;r++)
                         b[r]=c2[r];
                        for(r=0;r<f;r++)
                        {
                                for(j=r;j<f;j++)
                                {
                                        if(b[r]<b[j])
```

```c
                {
                        t=b[r];
                        b[r]=b[j];
                        b[j]=t;
                }
        }
}
for(r=0;r<f;r++)
{
        if(c2[r]==b[0])
        q[r]=p[i];
        printf("\t%d",q[r]);
}
printf("\n");
        }
    }
}
printf("\nThe no of page faults is %d",c);
}
```

BEST FIT ALGORITHM

```c
#include<stdio.h>

void main()
{
int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
static int barray[20],parray[20];
printf("\n\t\t\tMemory Management Scheme - Best Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of processes:");
scanf("%d",&np);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
   {
printf("Block no.%d:",i);
     scanf("%d",&b[i]);
   }
printf("\nEnter the size of the processes :-\n");
for(i=1;i<=np;i++)
   {
     printf("Process no.%d:",i);
     scanf("%d",&p[i]);
   }
for(i=1;i<=np;i++)
{
for(j=1;j<=nb;j++)
{
if(barray[j]!=1)
{
temp=b[j]-p[i];
if(temp>=0)
if(lowest>temp)
{
parray[i]=j;
lowest=temp;
}
}
}
fragment[i]=lowest;
barray[parray[i]]=1;
lowest=10000;
}
printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
for(i=1;i<=np && parray[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);
}
```

SEQ FILE ALLOCATION

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

void recurse(int files[]){
    int flag = 0, startBlock, len, j, k, ch;
    printf("Enter the starting block and the length of the files: ");
    scanf("%d%d", &startBlock, &len);
    for (j=startBlock; j<(startBlock+len); j++){
        if (files[j] == 0)
            flag++;
    }
    if(len == flag){
        for (int k=startBlock; k<(startBlock+len); k++){
            if (files[k] == 0){
                files[k] = 1;
                printf("%d\t%d\n", k, files[k]);
            }
        }
        if (k != (startBlock+len-1))
            printf("The file is allocated to the disk\n");
    }
    else
        printf("The file is not allocated to the disk\n");

    printf("Do you want to enter more files?\n");
    printf("Press 1 for YES, 0 for NO: ");
    scanf("%d", &ch);
    if (ch == 1)
        recurse(files);
    else
        exit(0);
    return;
}

int main()
{
int files[50];
for(int i=0;i<50;i++)
files[i]=0;
printf("Files Allocated are :\n");

recurse(files);
getch();
return 0;
}
```

SCAN DISK SCHEDULING

```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int choice,track,no_req,head,head1,distance;
int disc_req[100],finish[100];
void menu()
{
printf("\n\n*MENU*");
printf("\n1. Input data\n 2. SCAN \n 3. Exit");
printf("\n\n Enter your choice");
scanf("%d",&choice);
}
void input()
{
int i;
printf("Enter Total number of tracks");
scanf("%d",&track);
printf("Enter total number of disc requests");
scanf("%d",&no_req);
printf("\n Enter disc requests in FCFS order");
for(i=0;i<no_req;i++)
{
scanf("%d",&disc_req[i]);
}
printf("\n Enter current head position");
scanf("%d",&head1);
}
void sort()
{
int i,j,temp;
for(i=0;i<no_req;i++)
{
for(j=0;j<no_req;j++)
{
if(disc_req[i]<disc_req[j])
{
temp=disc_req[i];
disc_req[i]=disc_req[j];
disc_req[j]=temp;
}
}
}
}
void scan()
{
int index,dir;
int i;
distance=0;
head=head1;
printf("\n Enter the direction of head \n 1 - Towars higher
```

```c
disc(Right) \n 0 -towards lower disc(left)");
scanf("%d",&dir);
sort();
printf("\n Sorted Disc requests are: ");
for(i=0;i<no_req;i++)
{
printf(" %d",disc_req[i]);
}
i=0;
while(head>=disc_req[i])
{
index=i;
i++;
}
printf("\n index=%d",index);
printf("\n%d=>",head);
if(dir==1)
{
sort();
for(i=index+1;i<no_req;i++)
{
printf("%d=>",disc_req[i]);
distance+=abs(head-disc_req[i]);
head=disc_req[i];
}
distance+=abs(head-(track-1));
printf("%d=>",track-1);
head=track-1;
for(i=index;i>=0;i--)
{
printf("%d=>",disc_req[i]);
distance+=abs(head-disc_req[i]);
head=disc_req[i];
}
}
else
{
sort();
for(i=index;i>=0;i--)
{
printf("%d=>",disc_req[i]);
distance+=abs(head-disc_req[i]);
head=disc_req[i];
}
distance+=abs(head-0);
head=0;
printf("0=>");
for(i=index+1;i<no_req;i++)
{
printf("%d=>",disc_req[i]);
distance+=abs(head-disc_req[i]);
```

```c
head=disc_req[i];
}
}
printf("End");
printf("\n Total Distance Traversed=%d",distance);
}
int main()
{
while(1)
{
menu();
switch(choice)
{
case 1: input();
break;
case 2: scan();
break;
case 3: exit(0);
break;
default:
printf("\n Enter valid choice");
break;
}
}
return 0;
}
```