Operating System -2 Practicals

1.Write a C Menu driven Program to implement following functionality

a) Accept Available

b) Display Allocation, Max

c) Display the contents of need matrix

d) Display Available

```
#include<stdio.h>
#include<stdlib.h>
int allocation[20][20],max[20][20],available[20],need[20][20],safe[10],s=0;
int
finish[10],work[10],cnt=0,flag=0,temp=0;
int p,r,i,j,ch,index,req[10];
void check()
{
temp=0;
s=0;
for(i=0;i<p;i++) //Calculate need=max-allocation
for(j=0;j<r;j++)
need[i][j]=max[i][j]-allocation[i][j];
printf("\nAllocation Table is:\n");
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
printf("%d\t",allocation[i][j]);
printf("\n");
}
printf("\nNeed Table is:\n");
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
printf("%d\t",need[i][j]);
printf("\n");
}
for(i=0;i<p;i++)
finish[i]=0;
for(i=0;i<r;i++)
work[i]=available[i];
while(temp<2)
{
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
{
```

```c
}
if(finish[i]==0 && need[i][j]<=work[j])
flag=1;
else
{
flag=0;
break;
}
if(flag==1)
{
for(j=0;j<r;j++)
work[j]=work[j]+allocation[i][j];
finish[i]=1;
safe[s++]=i;
}
}
temp++;
}
flag=0;
for(i=0;i<p;i++)
{
}
if(finish[i]==0)
{
flag=1;
 //break;
}
if(flag==1)
{
printf("\nSystem is in Deadlock state");
}
else
{
printf("\nSystem is in Safe state");
printf("\nSafe Sequence is:");
for(i=0;i<p;i++)
printf("P%d\t",safe[i]);
}
}
void main()
{
printf("\n~~~~~~~~~~~~~~~~~~~~~~~BANKER'S
ALGORITHM~~~~~~~~~~~~~~~~~~~~~~~~~~");
printf("\n\nEnter the no of resources and processes: ");
scanf("%d%d",&r,&p);
printf("\nEnter the Allocation Table:\n");
for(i=0;i<p;i++) //Accept the allocation table
for(j=0;j<r;j++)
```

```c
scanf("%d",&allocation[i][j]);
printf("\nEnter the Max Table:\n");
for(i=0;i<p;i++) //Accept the max table
for(j=0;j<r;j++)
scanf("%d",&max[i][j]);
printf("\nEnter the vector Available :");
for(i=0;i<r;i++)
scanf("%d",&available[i]);
check();
printf("\nDo U want to add new request:(0/1)");
scanf("%d",&ch);
if(ch==0)
exit(1);
printf("\nEnter the process no:");
scanf("%d",&index);
printf("\nEnter the request:");
for(i=0;i<r;i++)
scanf("%d",&req[i]);
flag=0;
for(i=0;i<r;i++)
if(req[i]<=need[index][i])
flag=1;
else
flag=0;
if(flag==0)
{
printf("\nRequest can not be satisfied...");
exit(1);
}
for(i=0;i<r;i++)
if(req[i]<=available[i])
flag=1;
else
flag=0;
if(flag==0)
{
printf("\nRequest can not be satisfied...");
exit(1);
}
}
```

2.Write a program to simulate Linked file allocation method. Assume disk with n number of blocks. Give value of n as input. Randomly mark some block as allocated and accordingly maintain the list of free blocks Write menu driver program with menu options as mentioned below and implement each option.

a) Show Bit Vector

b) Create New File

c) Show Directory

d) Exit

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX 200
typedef struct dir
{
char fname[20];
int start; struct
dir *next;
}NODE;
NODE *first,*last;
int n,fb,bit[MAX];
void init()
{ int
i;
printf("Enter total no.of disk blocks:");
scanf("%d",&n);
fb
=
n;
for(i=0;i<10;i++)
{
int k = rand()%n;
if(bit[k]!=-2)
{
bit[k]=-2;
fb--;
}
}
}
void show_bitvector()
{
int i;
for(i=0;i<n;i++)
printf("%d ",bit[i]);
printf("\n");
}
```

```
void show_dir()
{
NODE *p;
int i;
printf("File\tChain\n");
p = first;
while(p!=NULL)
{
printf("%s\t",p->fname);
i = p->start; while(i!=-1)
{
printf("%d->",i);
i=bit[i];
}
printf("NULL\n");
p=p->next;
}
}
void create()
{
NODE *p; char
fname[20]; int
i,j,nob;
printf("Enter file name:");
scanf("%s",fname);
printf("Enter no.of blocks:");
scanf("%d",&nob);
if(nob>fb)
{
printf("Failed to create file %s\n",fname);
return;
}
for(i=0;i<n;i++)
{
if(bit[i]==0) break;
}
p = (NODE*)malloc(sizeof(NODE));
strcpy(p->fname,fname); p->start=i;
p->next=NULL;
if(first==NULL)
first=p; else
last->next=p;
last=p;
fb-=nob;
j=i+1; nob--
;
while(nob>0)
{
```

```
if(bit[j]==0)
{
bit[i]=j;
i=j;
nob--;
}
j++;
}
bit[i]=-1;
printf("File %s created successully.\n",fname);
}
void delete()
{
char fname[20];
NODE *p,*q; int
nob=0,i,j;
printf("Enter file name to be deleted:");
scanf("%s",fname);
p = q = first;
while(p!=NULL)
{
if(strcmp(p->fname,fname)==0)
break;
q=p;
p=p->next;
}
if(p==NULL)
{
printf("File %s not found.\n",fname);
return;
}
i = p->start;
while(i!=-1)
{
nob++;
j = i;
 i = bit[i]; bit[j] = 0;
}
fb+=nob;
if(p==first) first=first->next;
else if(p==last)
{
last=q; last->next=NULL;
}
else q->next = p->next;
free(p);
printf("File %s deleted successfully.\n",fname);
}
```

```
int main()
{ int
ch;
init();
while(1)
{
printf("1.Show bit vector\n");
printf("2.Create new file\n");
printf("3.Show directory\n");
printf("4.Delete file\n");
printf("5.Exit\n"); printf("Enter your choice (1-5):");
scanf("%d",&ch);
switch(ch)
{
case 1:
show_bitvector();
break; case 2:
create();
break; case
3:
show_dir();
break; case
4: delete();
break; case
5: exit(0);
}
}
return 0;
}
```

3.Write a simulation program for disk scheduling using **FCFS algorithm**. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of requests in the order in which it is served. Also display the total number of head moments.

55, 58, 39, 18, 90, 160, 150, 38, 184

Start Head Position: 50

```
#include<stdio.h>
int main()
{
int i,j,sum=0,n; int
ar[20],tm[20];
int disk;
```

```c
printf("enter number of location\t"); scanf("%d",&n);
printf("enter position of head\t"); scanf("%d",&disk);
printf("enter elements of disk queue\n");
for(i=0;i<n;i++)
{
scanf("%d",&ar[i]);
tm[i]=disk-ar[i];
if(tm[i]<0)
{
tm[i]=ar[i]-disk;
}
disk=ar[i];
sum=sum+tm[i];
}
/*for(i=0;i<n;i++)
{
printf("\n%d->",tm[i]);
} */
printf("\nmovement of total cylinders %d",sum);
return 0;
}
```

4.Write a simulation program for disk scheduling using **SSTF algorithm**. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of requests in the order in which it is served. Also display the total number of head moments.

80, 150, 60,135, 40, 35, 170

Starting Head Position: 70

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int RQ[100], i, n, TotalHeadMovement = 0, initial, count = 0;
    int visited[100] = {0}; // To keep track of visited requests
    printf("Enter the number of Requests: ");
    scanf("%d", &n);
    printf("Enter the Requests sequence: ");
    for(i = 0; i < n; i++)
        scanf("%d", &RQ[i]);
    printf("Enter initial head position: ");
    scanf("%d", &initial);
    printf("%d", initial);
    // SSTF Logic: Process requests in the shortest seek time first order
    while(count != n)
    {
        int min = 1000, index = -1, d;
        for(i = 0; i < n; i++)
        {
            if (!visited[i])  // Process only unvisited requests
            {
                d = abs(RQ[i] - initial);
                if (d < min)
                {
                    min = d;
                    index = i;
                }
            }
        }
        // If no valid index found, break
        if (index == -1)
            break;
        // Move head to the selected request
        visited[index] = 1;
        TotalHeadMovement += min;
        initial = RQ[index];
        printf(" --> %d", RQ[index]);
        count++;
    }
    printf("\nTotal head movement is %d\n", TotalHeadMovement);
    return 0;
}
```

5.Write a simulation program for disk scheduling using **SCAN algorithm**. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of requests in the order in which it is served. Also display the total number of head moments.

55, 58, 39, 18, 90, 160, 150, 38, 184

Start Head Position: 50

Direction: Right

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

```c
int main()
{
int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
printf("Enter the number of Requests\n"); scanf("%d",&n);
printf("Enter the Requests sequence\n");
for(i=0;i<n;i++)
scanf("%d",&RQ[i]);
printf("Enter initial head position\n");
scanf("%d",&initial);
printf("Enter total disk size\n"); scanf("%d",&size);
printf("Enter the head movement direction for high 1 and for low 0\n");
scanf("%d",&move);
// logic for Scan disk scheduling
/*logic for sort the request array */
for(i=0;i<n;i++)
{
for(j=0;j<n-i-1;j++)
{
if(RQ[j]>RQ[j+1])
{
int temp;
temp=RQ[j];
RQ[j]=RQ[j+1];
RQ[j+1]=temp;
}
}
}
int index;
for(i=0;i<n;i++)
{
if(initial<RQ[i])
{
index=i;
break;
}
}
// if movement is towards high value
if(move==1)
{
for(i=index;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
// last movement for max size
TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
initial = size-1;
for(i=index-1;i>=0;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
}
// if movement is towards low value
else
{
for(i=index-1;i>=0;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
// last movement for min size
TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
initial =0;
for(i=index;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
}
printf("Total head movement is %d",TotalHeadMoment);
return 0;
}
```

6.Write a simulation program for disk scheduling using **C-SCAN algorithm**. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of requests in the order in which it is served. Also display the total number of head moments.

80, 150, 60,135, 40, 35, 170

Starting Head Position: 70

Direction: Right

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

```
int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
printf("Enter the number of Requests\n"); scanf("%d",&n);
printf("Enter the Requests sequence\n");
for(i=0;i<n;i++)
scanf("%d",&RQ[i]);
printf("Enter initial head position\n");
scanf("%d",&initial);
printf("Enter total disk size\n"); scanf("%d",&size);
printf("Enter the head movement direction for high 1 and for low 0\n");
scanf("%d",&move);
// logic for C-Scan disk scheduling
/*logic for sort the request array */
for(i=0;i<n;i++)
{
for( j=0;j<n-i-1;j++)
{
if(RQ[j]>RQ[j+1])
{
int temp;
temp=RQ[j];
RQ[j]=RQ[j+1];
RQ[j+1]=temp;
}
}
}
int index;
for(i=0;i<n;i++)
{
if(initial<RQ[i])
{
index=i;
break;
}
}
// if movement is towards high value
if(move==1)
{
for(i=index;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
// last movement for max size
TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
/*movement max to min disk */
TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
```

```c
initial=0;
for( i=0;i<index;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
}
// if movement is towards low value
else
{
for(i=index-1;i>=0;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
// last movement for min size
TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
/*movement min to max disk */
TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
initial =size-1;
for(i=n-1;i>=index;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}
}
printf("Total head movement is %d",TotalHeadMoment);
return 0;
}
```