Slip – 1

Q.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

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	2	3	2	9	7	5	3	3	2
P1	4	0	0	5	2	2			
P2	5	0	4	1	0	4			
P3	4	3	3	4	4	4			
P4	2	2	4	6	5	5			

Ans →

Source Code –

```
#include<stdio.h>
int A[10][10], M[10][10], N[10][10];
int AV[10], Safe[10], Finish[10], R[10][10], nor, nop;
void AcceptData(intX[][10])
{
int i, j;
for (i = 0; i < nop; i++)
{
printf("P%d\n", i);
for (j = 0; j < nor; j++)
{
printf("%c:", 65 + j); scanf("%d", &X[i][j]);

}
}
}
void AcceptAvailable()
{
int i;
for (i = 0; i < nor; i++)
{
printf("%c:", 65 + i);
scanf("%d", &AV[i]);
}
}
void AcceptRequest(intR[][10])
{
int i;
for (i = 0; i < nor; i++)
{
printf("%c:", 65 + i);
scanf("%d", &R[i]);
}
}
void DisplayData()
{
```

```
int i, j; printf("\n\tAllocation\t\tMax\t\tNeed\n"); printf("\t");
for (i = 0; i < 3; i++)
{
for (j = 0; j < nor; j++)
{
printf("%4c", 65 + j);
}
printf("\t");
}
for (i = 0; i < nop; i++)
```

```
{
printf("\nP%d\t", i);
for (j = 0; j < nor; j++)
{
printf("%4d", A[i][j]);
}
printf("\t");
for (j = 0; j < nor; j++)
{
printf("%4d", M[i][j]);
}
printf("\t");
for (j = 0; j < nor; j++) printf("%4d", N[i][j]);
}
printf("\nAvailable\n"); for (j = 0; j < nor; j++)
{
printf("%4d", AV[j]);
}
}

void CalcNeed()
{
int i, j;
for (i = 0; i < nop; i++)
{
for (j = 0; j < nor; j++)
{
N[i][j] = M[i][j] - A[i][j];
}
}
}
int CheckNeed(intpno)

{

int i;
for (i = 0; i < nor; i++)
{
if (N[pno][i] > AV[i])
{
return 0;
}
}
return 1;
}
int main()
{
printf("\nEnter Number of Process :"); scanf("%d", &nop);
printf("\nEnter Number of Resources :"); scanf("%d", &nor);
printf("Enter Allocation\n"); AcceptData(A);
printf("Enter MAX Requirement\n"); AcceptData(M);
printf("Enter Availability\n"); AcceptAvailable();
CalcNeed(); DisplayData();
}
```

Output –

```
Enter Number of
Process :
Enter Number of     3
Resources :
Enter Allocation
P0        A:0       B:1       C:0
P1        A:2       B:0       C:0
P2        A:3       B:0       C:2
P3        A:2       B:1       C:1
P4 A:0 B:0 C:2
Enter MAX Requirement  A:7                   B:5                  C:3
P0
P1                     A:3                   B:2                  C:2
P2                     A:9                   B:0                  C:2
P3                     A:2                   B:2                  C:2
P4                     A:4                   B:3                  C:3
Enter Availability A:2 B:4 C:6

Allocation          Max                 Need
A      B      C      A      B      C      A      B      C
P0 0   1      0      7      5      3      7      4      3
P1 2   0      0      3      2      2      1      2      2
P2 3   0      2      9      0      2      6      0      0
P3 2   1      1      2      2      2      0      1      1
P4 0   0      2      4      3      3      4      3      1
Available
2 4 6
```

Q.2 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 request 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

Ans →

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100],i,n,TotalHeadMoment=0,initial;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
     scanf("%d",&RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d",&initial);
    // logic for FCFS disk scheduling
    for(i=0;i<n;i++)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }

    printf("Total head moment is %d",TotalHeadMoment);
    return 0;

}
```

Output

```
Enter the number of Request
8
Enter the Requests Sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Total head movement is 644
```

Slip – 2

Q.1 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.

- Show Bit Vector
- Create New File
- Show Directory
- Exit

Ans →

```
 #include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct file
{
char filename[20];
int filesize;
int startblock;
int endblock;
};
typedef struct file f;
struct node
{
int blockno;
struct node *next;
};
typedef struct node block;
block *freelist=NULL,*allocate=NULL,*lasta=NULL;
f f1[20];
int no=1,d,size,count=1,countfblock=0;
block *getblock()
{
block *temp;
temp=(block *)malloc(sizeof(block));
temp->blockno=no;
no++;
temp->next=NULL;
return(temp);
}

block * addblock(block *ptr)
{
block *temp;
temp=(block *)malloc(sizeof(block));
temp->blockno=ptr->blockno;
temp->next=NULL;
return(temp);
}
block *create()
{
block *temp,*last=NULL,*list=NULL;
int i;
for(i=0;i<d;i++)
{
temp=getblock();
if(list==NULL)
{
list=temp;
last=temp;
}
else
{
last->next=temp;
last=temp;
}
}
return(list);
}
block *createalloclist()
{
block *temp,*ptr=freelist,*prev;
int i;
f1[count].startblock=ptr->blockno;


for(i=0;i<f1[count].filesize && ptr!=NULL;i++)
{
temp=addblock(ptr);
f1[count].endblock=temp->blockno;
prev=ptr;
freelist=ptr->next;
ptr=ptr->next;
if(allocate==NULL)
{
allocate=temp;
lasta=temp;
}
else
{
lasta->next=temp;
lasta=temp;
}
}
return(allocate);
}

{
printf("\nEnter the file name:");
scanf("%s",&f1[count].filename);
printf("\nEnter file size in blocks:");
scanf("%d",&f1[count].filesize);
}

void displayfile()
{
int i;
printf("\nFile name\t\tFile size\t\tstart block\t\tEnd block");
for(i=1;i<=count;i++)
{
printf("\n%s",f1[i].filename);
printf("\t\t\t%d",f1[i].filesize);
printf("\t\t\t%d",f1[i].startblock);
printf("\t\t\t%d",f1[i].endblock);
}
}
int main()
{
int ch,result;
char fname[20];
printf("\nEnter the size of disk in blocks");
```

```
scanf("%d",&d);
freelist=create();
while(1)
{
printf("\n1: Allocate space for newly created file.");
printf("\n2: Show used and free space on disk.");
printf("\n3: Exit");
printf("\nEnter the choice");
scanf("%d",&ch);
switch(ch)
{
case 1:
acceptfile();
countfree(freelist);
if(countfblock>=f1[count].filesize)
{
allocate=createalloclist();
displayfile();
count++;
}
else
printf("\nNo sufficient space to allocate");
break;
case 2:
printf("\nFree list:");
displaylist(freelist);
printf("\nAllocated list: ");
displaylist(allocate);
break;
case 3:
exit(0);
}
}
```

Q.2 Write an MPI program to calculate sum of randomly generated 1000 numbers (stored in array) on a cluster.

Ans →

```
 #include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define ARRAY_SIZE 1000
int main(int argc, char *argv[]) {
int rank, size;
int i, sum = 0;
int data[ARRAY_SIZE];
int local_sum = 0;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
// Generate random data
if (rank == 0)
{
printf("Generating random data...\n");
for (i = 0; i < ARRAY_SIZE; i++)
{
data[i] = rand() % 100;
}
}
// Scatter data to all processes
MPI_Scatter(data, ARRAY_SIZE/size, MPI_INT, data, ARRAY_SIZE/size, MPI_INT, 0,
MPI_COMM_WORLD);
// Calculate local sum
for (i = 0; i < ARRAY_SIZE/size; i++)
{
local_sum += data[i];
}
// Reduce local sums to get the final sum
MPI_Reduce(&local_sum, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
// Print the final sum
if (rank == 0) {
printf("The sum is: %d\n", sum);
}
MPI_Finalize();
return 0
```

Slip – 3

Q.1 Write a C program to simulate Banker's algorithm for the purpose of deadlock avoidance. Consider the following snapshot of system, A, B, C and D is the resource type.

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

a) Calculate and display the content of need matrix?
b) Is the system in safe state? If display the safe sequence.

Ans →

```
 #include<stdio.h>
#include<stdlib.h>
int ind,A[10][10],M[10][10],N[10][10],Av[10],Safe[10],Finish[10],nor,nop,work[10],req[10][10];
void AcceptData(int X[][10])
{
int i,j;
for(i=0;i<nop;i++)
{
printf("P%d:\n",i);
for(j=0;j<nor;j++)
{
printf("%c:",65+j);
scanf("%d",&X[i][j]);
}
}
}
void AcceptAvailability()
{
int i;
for(i=0;i<nor;i++)
{
printf("%c",65+i);
scanf("%d",&Av[i]);
work[i]=Av[i];
}
}
void DisplayData()
{
int i,j;
printf("\n\tAllocation\t\tMax\t\tNeed\n");
printf("\t");
for(i=0;i<3;i++)
{
for(j=0;j<nor;j++)
printf("%4c",65+j);
printf("\t");
}
for(i=0;i<nop;i++)
{
printf("\nP%d\t",i);
for(j=0;j<nor;j++)
printf("%4d",A[i][j]);
printf("\t");
for(j=0;j<nor;j++)
printf("%4d",M[i][j]);
printf("\t");
for(j=0;j<nor;j++)
printf("%4d",N[i][j]);
}
printf("\nAvailable\n”);
for(j=0;j<nor;j++)
printf("%4d",work[j]);
}
void CalcNeed()
{
int i,j;
for(i=0;i<nop;i++)
for(j=0;j<nor;j++)
N[i][j]=M[i][j]-A[i][j];
}
void Resource_Request(int no)
{
int i,f11=0,f12=0;
for(i=0;i<nor;i++)
{
if(req[no][i]<=N[no][i])
f11=1;
else
```

```
f11=0;
}
if(f11==0)
{
printf("\n Error!Process has exceeded its maximum claim");
exit(0);
}
if(f11==1)
{
for(i=0;i<nor;i++)
{
if(req[no][i]<=work[i])
f12=1;
else
f12=0;
}
if(f12==0)
{
printf("\n Process has to wait for resources");
exit(0);
}
}
if(f11==1 && f12==1)
{
for(i=0;i<nor;i++)
{
work[i]=work[i]-req[no][i];
A[no][i]=A[no][i]+req[no][i];
N[no][i]=N[no][i]-req[no][i];
}
}
}
int checkNeed(int pno)
{
int i;
for(i=0;i<nor;i++)
if(N[pno][i]>work[i])
return(0);
return(1);
}
void Banker()
{
int i=0,j=0,k=0,flag=0;
while(flag<2)
{
if(!Finish[i])
{
printf("\nNeed%d(",i);
for(j=0;j<nor;j++)
printf("%d",N[i][j]);
if(!checkNeed(i))
{
printf("\b)>Work");
for(j=0;j<nor;j++)
printf("%d",work[j]);
printf("\b)");
printf("\nNeed Cannot be satisfied,consider next process");
}
else
{
printf("b)<=Work(");
for(j=0;j<nor;j++)
printf("%d,",work[j]);
printf("\b)");
printf("\nNeed can be satisfied,so allocate required resources");
printf("\nWork(%d)=",i);
for(j=0;j<nor;j++)
{
work[j]+=A[i][j];
}
```

```
for(j=0;j<nor;j++)
printf("%4d",work[j]);
printf("\nAfter P%d terminates it will release all its resources\n",i);
Safe[k++]=i;
Finish[i]=1;
}
}
if((i+1)%nop==0)
flag++;
i=(i+1)%nop;
}
if(k==nop)
{
printf("\nSystem is in safe state...");
printf("\nSafe Sequence:");
for(i=0;i<k;i++)
printf("P%d->",Safe[i]);
printf("\b\b");
}
else
{
printf("\nSystem is in not safe state...");
}
}
int main()
{
int i;
printf("\nEnter no of processes & No of Resources:");
scanf("%d%d",&nop,&nor);
printf("Enter Allocation\n");
AcceptData(A);
printf("Enter Max Requirement\n");
AcceptData(M);
printf("Enter Availability\n");
AcceptAvailability();
CalcNeed();
DisplayData();
Banker();
printf("\n Enter Process number from which request arrives:");
scanf("%d",&ind);
printf("\nEnter request for process%d\n",ind);
for(i=0;i<nor;i++)
{
printf("%c",65+i);
scanf("%d",&req[ind][i]);
}
for(i=0;i<nop;i++)
Finish[i]=0;
for(i=0;i<nor;i++)
work[i]=Av[i];
Resource_Request(ind);
Banker();
return(0);
}
```

✓ Q 2 Write an MPI program to calculate sum and average of randomly generated 1000 numbers (stored in array) on a cluster

Ans →

```
 #include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define ARRAY_SIZE 1000
int main(int argc, char *argv[]) {
int rank, size;
int i, sum = 0;
int data[ARRAY_SIZE];
int local_sum = 0;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

// Generate random data
if (rank == 0)
{
printf("Generating random data...\n");
for (i = 0; i < ARRAY_SIZE; i++)
{
data[i] = rand() % 100;
}
}
// Scatter data to all processes
MPI_Scatter(data, ARRAY_SIZE/size, MPI_INT, data, ARRAY_SIZE/size, MPI_INT, 0,
MPI_COMM_WORLD);
// Calculate local sum
for (i = 0; i < ARRAY_SIZE/size; i++)
{
local_sum += data[i];
}
// Reduce local sums to get the final sum
MPI_Reduce(&local_sum, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
// Print the final sum
if (rank == 0) {
printf("The sum is: %d\n", sum);
}
MPI_Finalize();
return 0;
}
```

Slip – 4

Q.1 Implement the Menu driven Banker's algorithm for accepting Allocation, Max from user.
a) Accept Available
b) Display Allocation, Max
c) Find Need and display It,
d) Display Available
Consider the system with 3 resources types A,B, and C with 7,2,6 instances respectively.
Consider the following snapshot:

Process	Allocation			Request		
	A	B	C	A	B	C
P0	0	1	0	0	0	0
P1	4	0	0	5	2	2

P2	5	0	4	1	0	4
P3	4	3	3	4	4	4
P4	2	2	4	6	5	5

Ans →

```
 #include<stdio.h>
int A[10][10],M[10][10],N[10][10],Av[10],nor,nop;
void AcceptData(int X[][10])
{
int i,j;
for(i=0;i<nop;i++)
{
printf("P%d\n",i);
for(j=0;j<nor;j++)
{
printf("%c: ",65+j);
scanf("%d",&X[i][j]);
}
}
}
void AcceptAvailability()
{
int i;
for(i=0;i<nor;i++)
{
printf("%c: ",65+i);
scanf("%d",&Av[i]);
}
}

void DisplayData()
{
int i,j;
printf("\n\tAllocation\t\tMax\t\tNeed\n");
printf("\t");
for(i=0;i<3;i++)
{
for(j=0;j<nor;j++)
printf("%4c",65+j);
printf("\t");
}
for(i=0;i<nop;i++)
{
printf("\nP%d\t",i);
for(j=0;j<nor;j++)
printf("%4d",A[i][j]);
printf("\t");
for(j=0;j<nor;j++)
printf("%4d",M[i][j]);
printf("\t");
for(j=0;j<nor;j++)
printf("%4d",N[i][j]);
}
printf("\nAvailable\n");
for(j=0;j<nor;j++)
printf("%4d",Av[j]);
}
```

```
void CalcNeed()
{
int i,j;
for(i=0;i<nop;i++)
for(j=0;j<nor;j++)
N[i][j] = M[i][j] - A[i][j];
}
void main()
{
printf("\nEnter No.of Processes & No.of Resources: ");
scanf("%d %d",&nop,&nor);
printf("Enter Allocation\n");
AcceptData(A);
printf("Enter Max Requirement\n");

AcceptData(M);
printf("Enter Availability\n");
AcceptAvailability();
CalcNeed();
DisplayData();
}
```

Q.2 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 request in the order in which it is served. Also display the total number of head moments.
86, 147, 91, 170, 95, 130, 102, 70
Starting Head position= 125
Direction: Left.

Ans →

```
#include <stdio.h>

#include <math.h>

int main()

{

   int queue[20], n, head, i, j, k, seek = 0, max, diff, temp, queue1[20],

   queue2[20], temp1 = 0, temp2 = 0;

   float avg;

   printf("Enter the max range of disk\n");

   scanf("%d", &max);

   printf("Enter the initial head position\n");

   scanf("%d", &head);

   printf("Enter the size of queue request\n");

   scanf("%d", &n);

   printf("Enter the queue of disk positions to be read\n");

   for (i = 1; i <= n; i++)

   {

      scanf("%d", &temp);

      if (temp >= head)
```

```
    {
        queue1[temp1] = temp;
        temp1++;
    }
    else
    {
        queue2[temp2] = temp;
        temp2++;
    }
}
for (i = 0; i < temp1 - 1; i++)
{
    for (j = i + 1; j < temp1; j++)
    {
        if (queue1[i] > queue1[j])
        {
            temp = queue1[i];
            queue1[i] = queue1[j];
            queue1[j] = temp;
        }
    }
}
for (i = 0; i < temp2 - 1; i++)
{
    for (j = i + 1; j < temp2; j++)
    {
        if (queue2[i] < queue2[j])
        {
            temp = queue2[i];
            queue2[i] = queue2[j];
            queue2[j] = temp;
        }
    }
}
for (i = 1, j = 0; j < temp1; i++, j++)
    queue[i] = queue1[j];
queue[i] = max;
```

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

Slip – 5

Q.1 Consider a system with 'm' processes and 'n' resource types. Accept number of instances for every resource type. For each process accept the allocation and maximum requirement matrices. Write a program to display the contents of need matrix and to check if the given request of a process can be granted immediately or not

Ans →

```
 #include<stdio.h>
#include<stdlib.h>
int ind,A[10][10],M[10][10],N[10][10],Av[10],Safe[10],Finish[10],nor,nop,work[10],req[10][10];
void AcceptData(int X[][10])
{
int i,j;
for(i=0;i<nop;i++)
{
printf("P%d:\n",i);
for(j=0;j<nor;j++)
{
printf("%c:",65+j);
scanf("%d",&X[i][j]);
}
}
}
void AcceptAvailability()
{
int i;
for(i=0;i<nor;i++)
{
printf("%c",65+i);
scanf("%d",&Av[i]);
work[i]=Av[i];
}
}
void DisplayData()
{
int i,j;
printf("\n\tAllocation\t\tMax\t\tNeed\n");
printf("\t");
for(i=0;i<3;i++)
{
for(j=0;j<nor;j++)
printf("%4c",65+j);
printf("\t");
}
for(i=0;i<nop;i++)
{
printf("\nP%d\t",i);
for(j=0;j<nor;j++)
printf("%4d",A[i][j]);
printf("\t");
for(j=0;j<nor;j++)
printf("%4d",M[i][j]);
printf("\t");
for(j=0;j<nor;j++)
printf("%4d",N[i][j]);
}
printf("\nAvailable\n");
for(j=0;j<nor;j++)
printf("%4d",work[j]);
}
void CalcNeed()
{
int i,j;
for(i=0;i<nop;i++)
for(j=0;j<nor;j++)
N[i][j]=M[i][j]-A[i][j];
}
void Resource_Request(int no)
{
int i,f11=0,f12=0;
for(i=0;i<nor;i++)
{
if(req[no][i]<=N[no][i])
```

```
f11=1;
else
f11=0;
}
if(f11==0)
{
printf("\n Error!Process has exceeded its maximum claim");
exit(0);
}
if(f11==1)
{
for(i=0;i<nor;i++)
{
if(req[no][i]<=work[i])
f12=1;
else
f12=0;
}
if(f12==0)
{
printf("\n Process has to wait for resources");
exit(0);
}
}
if(f11==1 && f12==1)
{
for(i=0;i<nor;i++)
{
work[i]=work[i]-req[no][i];
A[no][i]=A[no][i]+req[no][i];
N[no][i]=N[no][i]-req[no][i];
}
}
}
int checkNeed(int pno)
{
int i;
for(i=0;i<nor;i++)
if(N[pno][i]>work[i])
return(0);
return(1);
}
void Banker()
{
int i=0,j=0,k=0,flag=0;
while(flag<2)
{
if(!Finish[i])
{
printf("\nNeed%d(",i);
for(j=0;j<nor;j++)
printf("%d",N[i][j]);
if(!checkNeed(i))
{
printf("\b)>Work");
for(j=0;j<nor;j++)
printf("%d",work[j]);
printf("\b)");
printf("\nNeed Cannot be satisfied,consider next process");
}
else
{
printf("b)<=Work(");
for(j=0;j<nor;j++)
printf("%d,",work[j]);
printf("\b)");
printf("\nNeed can be satisfied,so allocate required resources");
printf("\nWork(%d)=",i);
for(j=0;j<nor;j++)
{
```

```
work[j]+=A[i][j];
}
for(j=0;j<nor;j++)
printf("%4d",work[j]);
printf("\nAfter P%d terminates it will release all its resources\n",i);
Safe[k++]=i;
Finish[i]=1;
}
}
if((i+1)%nop==0)
flag++;
i=(i+1)%nop;
}
if(k==nop)
{
printf("\nSystem is in safe state...");
printf("\nSafe Sequence:");
for(i=0;i<k;i++)
printf("P%d->",Safe[i]);
printf("\b\b");
}
else
{
printf("\nSystem is in not safe state...");
}
}
int main()
{
int i;
printf("\nEnter no of processes & No of Resources:");
scanf("%d%d",&nop,&nor);
printf("Enter Allocation\n");
AcceptData(A);
printf("Enter Max Requirement\n");
AcceptData(M);
printf("Enter Availability\n");
AcceptAvailability();
CalcNeed();
DisplayData();
Banker();
printf("\n Enter Process number from which request arrives:");
scanf("%d",&ind);
printf("\nEnter request for process%d\n",ind);
for(i=0;i<nor;i++)
{
printf("%c",65+i);
scanf("%d",&req[ind][i]);
}
for(i=0;i<nop;i++)
Finish[i]=0;
for(i=0;i<nor;i++)
work[i]=Av[i];
Resource_Request(ind);
Banker();
return(0);
}
```

/*output:

Q.2 Write an MPI program to find the max number from randomly generated 1000 numbers (stored in array) on a cluster (Hint: Use MPI_Reduce)

Ans →

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

```
#define ARRAY_SIZE 1000
int main(int argc, char *argv[]) {
int rank, size;
int i, min, max;
int data[ARRAY_SIZE];
int local_min, local_max;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
// Generate random data on root process
if (rank == 0) {
printf("Generating random data...\n");
for (i = 0; i < ARRAY_SIZE; i++) {
data[i] = rand() % 100;
}
}
// Scatter data to all processes
MPI_Scatter(data, ARRAY_SIZE/size, MPI_INT, data, ARRAY_SIZE/size, MPI_INT, 0, MPI_COMM_WORLD);
// Calculate local min and max
local_min = data[0];
local_max = data[0];
for (i = 1; i < ARRAY_SIZE/size; i++) {
if (data[i] < local_min) {
local_min = data[i];
}
if (data[i] > local_max) {
local_max = data[i];
}
}
// Reduce local min and max to get the final min and max
MPI_Reduce(&local_min, &min, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);
MPI_Reduce(&local_max, &max, 1, MPI_INT, MPI_MAX, 0, MPI_COMM_WORLD);
// Print the final min and max on root process
if (rank == 0) {
printf("The minimum number is: %d\n", min);
printf("The maximum number is: %d\n", max);
}
MPI_Finalize();
return 0;
}
```

Slip – 6

Q.1 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.

- Show Bit Vector
- Create New File
- Show Directory
- Exit

Ans →

```
 #include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct file
{
char filename[20];
int filesize;
int startblock;
int endblock;
};
typedef struct file f;
struct node
{
int blockno;
struct node *next;
};
typedef struct node block;
block *freelist=NULL,*allocate=NULL,*lasta=NULL;
f f1[20];
int no=1,d,size,count=1,countfblock=0;
block *getblock()
{
block *temp;
temp=(block *)malloc(sizeof(block));
temp->blockno=no;
no++;
temp->next=NULL;
return(temp);
}
block * addblock(block *ptr)
{
block *temp;
temp=(block *)malloc(sizeof(block));
temp->blockno=ptr->blockno;
temp->next=NULL;
return(temp);
}
block *create()
{
block *temp,*last=NULL,*list=NULL;
int i;
for(i=0;i<d;i++)
{
temp=getblock();
if(list==NULL)
{
list=temp;
last=temp;
}
else
{
last->next=temp;
last=temp;
}
}
return(list);
}
block *createalloclist()
{
block *temp,*ptr=freelist,*prev;
int i;
f1[count].startblock=ptr->blockno;
for(i=0;i<f1[count].filesize && ptr!=NULL;i++)
{
temp=addblock(ptr);
f1[count].endblock=temp->blockno;
prev=ptr;
freelist=ptr->next;
ptr=ptr->next;
if(allocate==NULL)
{
allocate=temp;
lasta=temp;
}
else
{
lasta->next=temp;
lasta=temp;
}
}
return(allocate);
}

void displaylist(block *list1)
{
block *ptr;
for(ptr=list1;ptr!=NULL;ptr=ptr->next)
printf("%d->",ptr->blockno);
}
void countfree(block *list1)
{
block *ptr;
for(ptr=list1;ptr->next!=NULL;ptr=ptr->next)
countfblock++;
}
void acceptfile()
{
```

```
printf("\nEnter the file name:");
scanf("%s",&f1[count].filename);
printf("\nEnter file size in blocks:");
scanf("%d",&f1[count].filesize);
}
void displayfile()
{
int i;
printf("\nFile name\t\tFile size\t\tstart block\t\tEnd block");
for(i=1;i<=count;i++)
{
printf("\n%s",f1[i].filename);
printf("\t\t\t%d",f1[i].filesize);
printf("\t\t\t%d",f1[i].startblock);
printf("\t\t\t%d",f1[i].endblock);
}
}
int main()
{
int ch,result;
char fname[20];
printf("\nEnter the size of disk in blocks");
scanf("%d",&d);
freelist=create();
while(1)
{
printf("\n1: Allocate space for newly created file.");
printf("\n2: Show used and free space on disk.");
printf("\n3: Exit");
printf("\nEnter the choice");
scanf("%d",&ch);
switch(ch)
{
case 1:
acceptfile();
countfree(freelist);
if(countfblock>=f1[count].filesize)
{
allocate=createalloclist();
displayfile();
count++;
}
else
printf("\nNo sufficient space to allocate");
break;
case 2:
printf("\nFree list:");
displaylist(freelist);
printf("\nAllocated list: ");
displaylist(allocate);
break;
case 3:
exit(0);
}
}
}
```

Q.2 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 request 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

Ans →

```
#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);
        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;
}
```

Slip – 7

Q.1 Consider the following snapshot of the system.

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	2	0	0	1	4	2	1	2	3	3	2	1
P1	3	1	2	1	5	2	5	2				
P2	2	1	0	3	2	3	1	6				
P3	1	3	1	2	1	4	2	4				
P4	1	4	3	2	3	6	6	5				

Using Resource –Request algorithm to Check whether the current system is in safe state or not

Ans →

```
#include<stdio.h>
#include<stdlib.h>
int ind,A[10][10],M[10][10],N[10][10],Av[10],Safe[10],Finish[10],nor,nop,work[10],req[10][10];
void AcceptData(int X[][10])
{
int i,j;
for(i=0;i<nop;i++)
{
printf("P%d:\n",i);
```

```
for(j=0;j<nor;j++)
{
printf("%c:",65+j);
scanf("%d",&X[i][j]);
}
}
}
void AcceptAvailability()
{
int i;
for(i=0;i<nor;i++)
{
printf("%c",65+i);
scanf("%d",&Av[i]);
work[i]=Av[i];
}
}
void DisplayData()
{
int i,j;
printf("\n\tAllocation\t\tMax\t\tNeed\n");
printf("\t");
for(i=0;i<3;i++)
{
for(j=0;j<nor;j++)
printf("%4c",65+j);
printf("\t");
}
for(i=0;i<nop;i++)
{
printf("\nP%d\t",i);
for(j=0;j<nor;j++)
printf("%4d",A[i][j]);
printf("\t");
for(j=0;j<nor;j++)
printf("%4d",M[i][j]);
printf("\t");
for(j=0;j<nor;j++)
printf("%4d",N[i][j]);
}
printf("\nAvailable\n”);
for(j=0;j<nor;j++)
printf("%4d",work[j]);
}
void CalcNeed()
{
int i,j;
for(i=0;i<nop;i++)
for(j=0;j<nor;j++)
N[i][j]=M[i][j]-A[i][j];
}
void Resource_Request(int no)
{
int i,f11=0,f12=0;
for(i=0;i<nor;i++)
{
if(req[no][i]<=N[no][i])
f11=1;
else
f11=0;
}
if(f11==0)
{
printf("\n Error!Process has exceeded its maximum claim");
exit(0);
```

```
}
if(f11==1)
{
for(i=0;i<nor;i++)
{
if(req[no][i]<=work[i])
f12=1;
else
f12=0;
}
if(f12==0)
{
printf("\n Process has to wait for resources");
exit(0);
}
}
if(f11==1 && f12==1)
{
for(i=0;i<nor;i++)
{
work[i]=work[i]-req[no][i];
A[no][i]=A[no][i]+req[no][i];
N[no][i]=N[no][i]-req[no][i];
}
}
}
int checkNeed(int pno)
{
int i;
for(i=0;i<nor;i++)
if(N[pno][i]>work[i])
return(0);
return(1);
}
void Banker()
{
int i=0,j=0,k=0,flag=0;
while(flag<2)
{
if(!Finish[i])
{
printf("\nNeed%d(",i);
for(j=0;j<nor;j++)
printf("%d",N[i][j]);
if(!checkNeed(i))
{
printf("\b)>Work");
for(j=0;j<nor;j++)
printf("%d",work[j]);
printf("\b)");
printf("\nNeed Cannot be satisfied,consider next process");
}
else
{
printf("b)<=Work(");
for(j=0;j<nor;j++)
printf("%d,",work[j]);
printf("\b)");
printf("\nNeed can be satisfied,so allocate required resources");
printf("\nWork(%d)=",i);
for(j=0;j<nor;j++)
{
work[j]+=A[i][j];
}
for(j=0;j<nor;j++)
```

```
printf("%4d",work[j]);
printf("\nAfter P%d terminates it will release all its resources\n",i);
Safe[k++]=i;
Finish[i]=1;
}
}
if((i+1)%nop==0)
flag++;
i=(i+1)%nop;
}
if(k==nop)
{
printf("\nSystem is in safe state...");
printf("\nSafe Sequence:");
for(i=0;i<k;i++)
printf("P%d->",Safe[i]);
printf("\b\b");
}
else
{
printf("\nSystem is in not safe state...");
}
}
int main()
{
int i;
printf("\nEnter no of processes & No of Resources:");
scanf("%d%d",&nop,&nor);
printf("Enter Allocation\n");
AcceptData(A);
printf("Enter Max Requirement\n");
AcceptData(M);
printf("Enter Availability\n");
AcceptAvailability();
CalcNeed();
DisplayData();
Banker();
printf("\n Enter Process number from which request arrives:");
scanf("%d",&ind);
printf("\nEnter request for process%d\n",ind);
for(i=0;i<nor;i++)
{
printf("%c",65+i);
scanf("%d",&req[ind][i]);
}
for(i=0;i<nop;i++)
Finish[i]=0;
for(i=0;i<nor;i++)
work[i]=Av[i];
Resource_Request(ind);
Banker();
return(0);
}
```

Q.2 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 request in the order in which it is served. Also display the total number of head moments.
82, 170, 43, 140, 24, 16, 190
Starting Head Position: 50
Direction: Right.

Ans →

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

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

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

Slip – 8

Q.1 Write a program to simulate Contiguous 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 above and implement each option.

- Show Bit Vector
- Create New File
- Show Directory
- Exit

Ans →

```
 #include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct freelist
{
int start;
int size;
struct freelist *next;
};
struct usedlist
{
char fname[10];
int fstart;
int fsize;
};
struct freelist *head=NULL,*new=NULL,*temp,*prev,*temp1;
struct usedlist dir_ent[10];
int dir_index=0;
main()
{
int ch,i;
char filename[10];
create();
do

{
printf("\n*****menu*******\n");
printf("1.Create file\n");
printf("2. Show Free and Used Block list\n");
printf("3. exit\n");
printf("Enter your choice: ");
scanf("%d",&ch);
switch(ch)
{
case 1: allocate();
break;
case 2: printf("\nThe free list is");
printf("\nStartBlock\tSize\n");
for (temp=head;temp!=NULL; temp=temp->next)
{
printf("%d",temp->start);
printf("\t%d\n",temp->size);
}
printf("The used list is");
printf("\nFilename\tStart\tLength\n");
for(i=0;i<dir_index;i++)
{
printf("%s\t%d\t\t%d\n",dir_ent[i].fname,dir_ent[i].fstart,dir_ent[i].fsize);
}
break;
case 3: exit(0);
break;

}
}while (ch!=3);
}
create()
{
```

```
int no_of_blocks;
printf("Enter number of blocks\n");
scanf("%d",&no_of_blocks);
new = (struct freelist*)malloc(sizeof(struct freelist));
head=new;
new->start=0;
new->size=no_of_blocks;
new->next=NULL;
}
allocate()
{
int s, allocated=0;
char filename[10];
printf("enter file name \n");
scanf("%s",filename);
printf("enter size of a file in blocks\n");
scanf("%d",&s);
for(temp=head;temp!=NULL;)
{
if(temp->size < s)
temp=temp->next;
else

{
//temp->size-=s;
strcpy(dir_ent[dir_index].fname,filename);
dir_ent[dir_index].fstart=temp->start;
temp->start+=s;
dir_ent[dir_index].fsize=s;
dir_index++;
allocated=1;
break;
}
if (temp==NULL && allocated==0)
printf("Disk space not available\n");
}
}
```

Q.2 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 request in the order in which it is served. Also display the total number of head moments. 186, 89, 44, 70, 102, 22, 51, 124 Start Head Position: 70

Ans →

```
 #include<stdio.h>
#include<stdlib.h>
int main()
{
int RQ[100],i,n,TotalHeadMovement=0,initial,count=0;
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);
//sstf disk scheduling
```

```
while(count!=n)
{
int min=1000,d,index;
for(i=0;i<n;i++)
{
d=abs(RQ[i]-initial);
if(min>d)
{
min=d;
index=i;
}
}
TotalHeadMovement=TotalHeadMovement+min;
initial=RQ[index];
// 1000 is for max
// you can use any number
RQ[index]=1000;
count++;
}
printf("Total head movement is %d",TotalHeadMovement);
return 0
```

Slip – 9

Q.1. Consider the following snapshot of system, A, B, C, D is the resource type.

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

Using Resource –Request algorithm to Check whether the current system is in safe state or not .
Ans →

```
#include<stdio.h>
#include<stdlib.h>
int ind,A[10][10],M[10][10],N[10][10],Av[10],Safe[10],Finish[10],nor,nop,work[10],req[10][10];
void AcceptData(int X[][10])
{
int i,j;
for(i=0;i<nop;i++)
{
printf("P%d:\n",i);
for(j=0;j<nor;j++)
{
printf("%c:",65+j);
scanf("%d",&X[i][j]);
}
}
}
void AcceptAvailability()
{
int i;
for(i=0;i<nor;i++)
{
printf("%c",65+i);
scanf("%d",&Av[i]);
```

```
work[i]=Av[i];
}
}
void DisplayData()
{
int i,j;
printf("\n\tAllocation\t\tMax\t\tNeed\n");
printf("\t");
for(i=0;i<3;i++)
{
for(j=0;j<nor;j++)
printf("%4c",65+j);
printf("\t");
}
for(i=0;i<nop;i++)
{
printf("\nP%d\t",i);
for(j=0;j<nor;j++)
printf("%4d",A[i][j]);
printf("\t");
for(j=0;j<nor;j++)
printf("%4d",M[i][j]);
printf("\t");
for(j=0;j<nor;j++)
printf("%4d",N[i][j]);
}
printf("\nAvailable\n”);
for(j=0;j<nor;j++)
printf("%4d",work[j]);
}
void CalcNeed()
{
int i,j;
for(i=0;i<nop;i++)
for(j=0;j<nor;j++)
N[i][j]=M[i][j]-A[i][j];
}
void Resource_Request(int no)
{
int i,f11=0,f12=0;
for(i=0;i<nor;i++)
{
if(req[no][i]<=N[no][i])
f11=1;
else
f11=0;
}
if(f11==0)
{
printf("\n Error!Process has exceeded its maximum claim");
exit(0);
}
if(f11==1)
{
for(i=0;i<nor;i++)
{
if(req[no][i]<=work[i])
f12=1;
else
f12=0;
}
if(f12==0)
{
printf("\n Process has to wait for resources");
exit(0);
}
}
if(f11==1 && f12==1)
{
for(i=0;i<nor;i++)
{
work[i]=work[i]-req[no][i];
A[no][i]=A[no][i]+req[no][i];
N[no][i]=N[no][i]-req[no][i];
}
}
}
int checkNeed(int pno)
{
int i;
for(i=0;i<nor;i++)
if(N[pno][i]>work[i])
return(0);
return(1);
}
void Banker()
{
int i=0,j=0,k=0,flag=0;
while(flag<2)
{
if(!Finish[i])
{
printf("\nNeed%d(",i);
for(j=0;j<nor;j++)
printf("%d",N[i][j]);
if(!checkNeed(i))
{
printf("\b)>Work");
for(j=0;j<nor;j++)
printf("%d",work[j]);
printf("\b)");
printf("\nNeed Cannot be satisfied,consider next process");
}
else
{
printf("b)<=Work(");
for(j=0;j<nor;j++)
printf("%d,",work[j]);
printf("\b)");
printf("\nNeed can be satisfied,so allocate required resources");
printf("\nWork(%d)=",i);
for(j=0;j<nor;j++)
{
work[j]+=A[i][j];
}
for(j=0;j<nor;j++)
printf("%4d",work[j]);
printf("\nAfter P%d terminates it will release all its resources\n",i);
Safe[k++]=i;
Finish[i]=1;
}
}
if((i+1)%nop==0)
flag++;
i=(i+1)%nop;
}
if(k==nop)
{
printf("\nSystem is in safe state...");
printf("\nSafe Sequence:");
```

```
for(i=0;i<k;i++)
printf("P%d->",Safe[i]);
printf("\b\b");
}
else
{
printf("\nSystem is in not safe state...");
}
}
int main()
{
int i;
printf("\nEnter no of processes & No of Resources:");
scanf("%d%d",&nop,&nor);
printf("Enter Allocation\n");
AcceptData(A);
printf("Enter Max Requirement\n");
AcceptData(M);
printf("Enter Availability\n");
AcceptAvailability();
CalcNeed();
DisplayData();
Banker();
printf("\n Enter Process number from which request arrives:");
scanf("%d",&ind);
printf("\nEnter request for process%d\n",ind);
for(i=0;i<nor;i++)
{
printf("%c",65+i);
scanf("%d",&req[ind][i]);
}
for(i=0;i<nop;i++)
Finish[i]=0;
for(i=0;i<nor;i++)
work[i]=Av[i];
Resource_Request(ind);
Banker();
return(0);
}
```

Q.2 Write a simulation program for disk scheduling using LOOK algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments. [15]
176, 79, 34, 60, 92, 11, 41, 114
Starting Head Position: 65
Direction: Left
Ans → ***No Answer***

Slip – 10

Q.1 Write an MPI program to calculate sum and average of randomly generated 1000 numbers (stored in array) on a cluster
Ans →

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

```
#include <mpi.h>
#define ARRAY_SIZE 1000
int main(int argc, char *argv[]) {
int rank, size;
int i, sum = 0;
int data[ARRAY_SIZE];
int local_sum = 0;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
// Generate random data
if (rank == 0)
{
printf("Generating random data...\n");
for (i = 0; i < ARRAY_SIZE; i++)
{
data[i] = rand() % 100;
}
}
// Scatter data to all processes

MPI_Scatter(data, ARRAY_SIZE/size, MPI_INT, data, ARRAY_SIZE/size, MPI_INT, 0, MPI_COMM_WORLD);
// Calculate local sum
for (i = 0; i < ARRAY_SIZE/size; i++)
{
local_sum += data[i];
}
// Reduce local sums to get the final sum
MPI_Reduce(&local_sum, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
// Print the final sum
if (rank == 0) {
printf("The sum is: %d\n", sum);
}
MPI_Finalize();
return 0;
}
```

Q.2 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 request in the order in which it is served. Also display the total number of head moments.

33, 99, 142, 52, 197, 79, 46, 65

Start Head Position: 72

Direction: Left.

Ans →

```
#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);
    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;
}
```

Slip – 11

Q.1 Write a C program to simulate Banker's algorithm for the purpose of deadlock avoidance. the following snapshot of system, A, B, C and D are the resource type.

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

Implement the following Menu.
a) Accept Available
b) Display Allocation, Max
c) Display the contents of need matrix
d) Display Available

Ans →

```
#include<stdio.h>
int A[10][10], M[10][10], N[10][10];
int AV[10], Safe[10], Finish[10], R[10][10], nor, nop;
void AcceptData(intX[][10])
{
int i, j;
for (i = 0; i < nop; i++)
{
printf("P%d\n", i);
for (j = 0; j < nor; j++)
{
printf("%c:", 65 + j); scanf("%d", &X[i][j]);

}
}
}
void AcceptAvailable()
{
int i;
for (i = 0; i < nor; i++)
{
printf("%c:", 65 + i);
scanf("%d", &AV[i]);
}
}
void AcceptRequest(intR[][10])
{
int i;
for (i = 0; i < nor; i++)
{
printf("%c:", 65 + i);
scanf("%d", &R[i]);
}
}
void DisplayData()
{
int i, j; printf("\n\tAllocation\t\tMax\t\tNeed\n"); printf("\t");
for (i = 0; i < 3; i++)
{
for (j = 0; j < nor; j++)
{
printf("%4c", 65 + j);
}
printf("\t");
}
for (i = 0; i < nop; i++)

{

printf("\nP%d\t", i);
for (j = 0; j < nor; j++)
{
printf("%4d", A[i][j]);
}
printf("\t");
for (j = 0; j < nor; j++)
{
printf("%4d", M[i][j]);
}
printf("\t");

for (j = 0; j < nor; j++) printf("%4d", N[i][j]);
}
printf("\nAvailable\n"); for (j = 0; j < nor; j++)
{
printf("%4d", AV[j]);
}
}

void CalcNeed()
{
int i, j;
```

```
for (i = 0; i < nop; i++)
{
for (j = 0; j < nor; j++)
{
N[i][j] = M[i][j] - A[i][j];
}
}
}
int CheckNeed(intpno)

{

int i;
for (i = 0; i < nor; i++)
{
if (N[pno][i] > AV[i])
{
return 0;
}
}
return 1;
}
int main()
{
printf("\nEnter Number of Process :"); scanf("%d", &nop);
printf("\nEnter Number of Resources :"); scanf("%d", &nor);
printf("Enter Allocation\n"); AcceptData(A);
printf("Enter MAX Requirement\n"); AcceptData(M);
printf("Enter Availability\n"); AcceptAvailable();
CalcNeed(); DisplayData();
```

Q.2 Write an MPI program to find the min number from randomly generated 1000 numbers (stored in array) on a cluster (Hint: Use MPI_Reduce)

Ans →

```
 #include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define ARRAY_SIZE 1000
int main(int argc, char *argv[]) {
int rank, size;
int i, min, max;
int data[ARRAY_SIZE];
int local_min, local_max;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
// Generate random data on root process
if (rank == 0) {
printf("Generating random data...\n");
for (i = 0; i < ARRAY_SIZE; i++) {
data[i] = rand() % 100;
}
}
```

```
// Scatter data to all processes
MPI_Scatter(data, ARRAY_SIZE/size, MPI_INT, data, ARRAY_SIZE/size, MPI_INT, 0, MPI_COMM_WORLD);
// Calculate local min and max
local_min = data[0];
local_max = data[0];
for (i = 1; i < ARRAY_SIZE/size; i++) {
if (data[i] < local_min) {
local_min = data[i];
}
if (data[i] > local_max) {
local_max = data[i];
}
}
// Reduce local min and max to get the final min and max
MPI_Reduce(&local_min, &min, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);
MPI_Reduce(&local_max, &max, 1, MPI_INT, MPI_MAX, 0, MPI_COMM_WORLD);
// Print the final min and max on root process
if (rank == 0) {
printf("The minimum number is: %d\n", min);
printf("The maximum number is: %d\n", max);
}
MPI_Finalize();
return 0;
```

Slip – 12

Q.1 Write an MPI program to calculate sum and average randomly generated 1000 numbers (stored in array) on a cluster.

Ans →

```
 int main(int argc, char *argv[]) {
int rank, size;
int i, sum = 0;
int data[ARRAY_SIZE];
int local_sum = 0;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
// Generate random data
if (rank == 0)
{
printf("Generating random data...\n");
for (i = 0; i < ARRAY_SIZE; i++)
{
data[i] = rand() % 100;
}
}
// Scatter data to all processes
MPI_Scatter(data, ARRAY_SIZE/size, MPI_INT, data, ARRAY_SIZE/size, MPI_INT, 0,
MPI_COMM_WORLD);
// Calculate local sum
for (i = 0; i < ARRAY_SIZE/size; i++)
{
local_sum += data[i];
}
// Reduce local sums to get the final sum
MPI_Reduce(&local_sum, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
// Print the final sum
if (rank == 0) {
printf("The sum is: %d\n", sum);
}
MPI_Finalize();
return 0;
}
```

Q.2 Write a simulation program for disk scheduling using C-LOOK algorithm. Accept total number of disk blocks, disk request string, and current head position from the user. Display the list of request in the order in which it is served. Also display the total number of head moments.

23, 89, 132, 42, 187, 69, 36, 55

Start Head Position: 40

Direction: Right

Ans → ***No Answer***

Slip – 13

Q.1 Write a C program to simulate Banker's algorithm for the purpose of deadlock avoidance. The following snapshot of system, A, B, C and D are the resource type.

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

a) Calculate and display the content of need matrix?
b) Is the system in safe state? If display the safe sequence.

Ans →

```
#include<stdio.h>
#include<stdlib.h>
int ind,A[10][10],M[10][10],N[10][10],Av[10],Safe[10],Finish[10],nor,nop,work[10],req[10][10];
void AcceptData(int X[][10])
{
int i,j;
for(i=0;i<nop;i++)
{
printf("P%d:\n",i);
for(j=0;j<nor;j++)
{
printf("%c:",65+j);
scanf("%d",&X[i][j]);
}
}
}
void AcceptAvailability()
{
int i;
for(i=0;i<nor;i++)
{
printf("%c",65+i);
scanf("%d",&Av[i]);
work[i]=Av[i];
}
}
void DisplayData()
{
int i,j;
printf("\n\tAllocation\t\tMax\t\tNeed\n");
printf("\t");
for(i=0;i<3;i++)
{
for(j=0;j<nor;j++)
printf("%4c",65+j);
printf("\t");
}
for(i=0;i<nop;i++)
{
printf("\nP%d\t",i);
for(j=0;j<nor;j++)
printf("%4d",A[i][j]);
printf("\t");
for(j=0;j<nor;j++)
printf("%4d",M[i][j]);
printf("\t");
for(j=0;j<nor;j++)
printf("%4d",N[i][j]);
}
printf("\nAvailable\n”);
for(j=0;j<nor;j++)
printf("%4d",work[j]);
}
void CalcNeed()
{
int i,j;
for(i=0;i<nop;i++)
for(j=0;j<nor;j++)
N[i][j]=M[i][j]-A[i][j];
}
void Resource_Request(int no)
{
int i,f11=0,f12=0;
for(i=0;i<nor;i++)
```

```
{
if(req[no][i]<=N[no][i])
f11=1;
else
f11=0;
}
if(f11==0)
{
printf("\n Error!Process has exceeded its maximum claim");
exit(0);
}
if(f11==1)
{
for(i=0;i<nor;i++)
{
if(req[no][i]<=work[i])
f12=1;
else
f12=0;
}
if(f12==0)
{
printf("\n Process has to wait for resources");
exit(0);
}
}
if(f11==1 && f12==1)
{
for(i=0;i<nor;i++)
{
work[i]=work[i]-req[no][i];
A[no][i]=A[no][i]+req[no][i];
N[no][i]=N[no][i]-req[no][i];
}
}
}
int checkNeed(int pno)
{
int i;
for(i=0;i<nor;i++)
if(N[pno][i]>work[i])
return(0);
return(1);
}
void Banker()
{
int i=0,j=0,k=0,flag=0;
while(flag<2)
{
if(!Finish[i])
{
printf("\nNeed%d(",i);
for(j=0;j<nor;j++)
printf("%d",N[i][j]);
if(!checkNeed(i))
{
printf("\b)>Work");
for(j=0;j<nor;j++)
printf("%d",work[j]);
printf("\b)");
printf("\nNeed Cannot be satisfied,consider next process");
}
else
{
printf("b)<=Work(");
for(j=0;j<nor;j++)
printf("%d,",work[j]);
printf("\b)");
printf("\nNeed can be satisfied,so allocate required resources");
printf("\nWork(%d)=",i);
for(j=0;j<nor;j++)
{
work[j]+=A[i][j];
}
for(j=0;j<nor;j++)
printf("%4d",work[j]);
printf("\nAfter P%d terminates it will release all its resources\n",i);
Safe[k++]=i;
Finish[i]=1;
}
}
if((i+1)%nop==0)
flag++;
i=(i+1)%nop;
}
if(k==nop)
{
printf("\nSystem is in safe state...");
printf("\nSafe Sequence:");
for(i=0;i<k;i++)
printf("P%d->",Safe[i]);
printf("\b\b");
}
else
{
printf("\nSystem is in not safe state...");
}
}
int main()
{
int i;
printf("\nEnter no of processes & No of Resources:");
scanf("%d%d",&nop,&nor);
printf("Enter Allocation\n");
AcceptData(A);
printf("Enter Max Requirement\n");
AcceptData(M);
printf("Enter Availability\n");
AcceptAvailability();
CalcNeed();
DisplayData();
Banker();
printf("\n Enter Process number from which request arrives:");
scanf("%d",&ind);
printf("\nEnter request for process%d\n",ind);
for(i=0;i<nor;i++)
{
printf("%c",65+i);
scanf("%d",&req[ind][i]);
}
for(i=0;i<nop;i++)
Finish[i]=0;
for(i=0;i<nor;i++)
work[i]=Av[i];
Resource_Request(ind);
Banker();
return(0);
}
```

Q.2 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 request in the order in which it is served. Also display the total number of head moments.
176, 79, 34, 60, 92, 11, 41, 114
Starting Head Position: 65
Direction: Left

Ans →

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

   for (i = 1, j = 0; j < temp1; i++, j++)
      queue[i] = queue1[j];
   queue[i] = max;
   for (i = temp1 + 2, j = 0; j < temp2; i++, j++)
      queue[i] = queue2[j];
   queue[i] = 0;
   queue[0] = head;
   for (j = 0; j <= n + 1; j++)
   {
      diff = abs(queue[j + 1] - queue[j]);
```

```
        seek += diff;
        printf("Disk head moves from %d to %d with seek %d\n", queue[j],
        queue[j + 1], diff);
    }
    printf("Total seek time is %d\n", seek);
    avg = seek / (float)n;
    printf("Average seek time is %f\n", avg);
    return 0;
}
```

Slip – 14

Q.1 Write a program to simulate Sequential (Contiguous) 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.

- Show Bit Vector
- Show Directory
- Delete File
- Exit

Ans →

```
 #include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct freelist
{
int start;
int size;
```

```
struct freelist *next;
};
struct usedlist
{
char fname[10];
int fstart;
int fsize;
};
struct freelist *head=NULL,*new=NULL,*temp,*prev,*temp1;
struct usedlist dir_ent[10];
int dir_index=0;
main()
{
int ch,i;
char filename[10];
create();
do

{
printf("\n*****menu*******\n");
printf("1.Create file\n");
printf("2. Show Free and Used Block list\n");
printf("3. exit\n");
printf("Enter your choice: ");
scanf("%d",&ch);
switch(ch)
{
case 1: allocate();
break;
case 2: printf("\nThe free list is");
printf("\nStartBlock\tSize\n");
for (temp=head;temp!=NULL; temp=temp->next)
{
printf("%d",temp->start);
printf("\t%d\n",temp->size);
}
printf("The used list is");
printf("\nFilename\tStart\tLength\n");
for(i=0;i<dir_index;i++)
{
printf("%s\t%d\t\t%d\n",dir_ent[i].fname,dir_ent[i].fstart,dir_ent[i].fsize);
}
break;
case 3: exit(0);
break;

}
}while (ch!=3);
}
create()
{
int no_of_blocks;
printf("Enter number of blocks\n");
scanf("%d",&no_of_blocks);
new = (struct freelist*)malloc(sizeof(struct freelist));
head=new;
new->start=0;
new->size=no_of_blocks;
new->next=NULL;
}
allocate()
{
```

```
int s, allocated=0;
char filename[10];
printf("enter file name \n");
scanf("%s",filename);
printf("enter size of a file in blocks\n");
scanf("%d",&s);
for(temp=head;temp!=NULL;)
{
if(temp->size < s)
temp=temp->next;
else

{
//temp->size-=s;
strcpy(dir_ent[dir_index].fname,filename);
dir_ent[dir_index].fstart=temp->start;
temp->start+=s;
dir_ent[dir_index].fsize=s;
dir_index++;
allocated=1;
break;
}
if (temp==NULL && allocated==0)
printf("Disk space not available\n");
}
}
```

Q.2 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 request 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

Ans → #include<stdio.h>

```
#include<stdlib.h>
int main()
{
int RQ[100],i,n,TotalHeadMovement=0,initial,count=0;
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);
//sstf disk scheduling

while(count!=n)
{
int min=1000,d,index;
for(i=0;i<n;i++)
{
d=abs(RQ[i]-initial);
if(min>d)
{
min=d;
index=i;
}
}
TotalHeadMovement=TotalHeadMovement+min;
initial=RQ[index];
// 1000 is for max
// you can use any number
RQ[index]=1000;
count++;
}
printf("Total head movement is %d",TotalHeadMovement);
return 0;
```

Slip – 15

Q.1 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.

- Show Bit Vector
- Create New File
- Show Directory
- Exit

Ans →

```
 #include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct file
{
char filename[20];
int filesize;
int startblock;
int endblock;
};
typedef struct file f;
struct node
{
int blockno;
struct node *next;
};
typedef struct node block;
block *freelist=NULL,*allocate=NULL,*lasta=NULL;
f f1[20];
int no=1,d,size,count=1,countfblock=0;
block *getblock()
{
block *temp;
temp=(block *)malloc(sizeof(block));
temp->blockno=no;
no++;
temp->next=NULL;
return(temp);
}

block * addblock(block *ptr)
{
block *temp;
temp=(block *)malloc(sizeof(block));
temp->blockno=ptr->blockno;
temp->next=NULL;
return(temp);
}
block *create()
{
block *temp,*last=NULL,*list=NULL;
int i;
```

```
for(i=0;i<d;i++)
{
temp=getblock();
if(list==NULL)
{
list=temp;
last=temp;
}
else
{
last->next=temp;
last=temp;
}
}
return(list);
}
block *createalloclist()
{
block *temp,*ptr=freelist,*prev;
int i;
f1[count].startblock=ptr->blockno;

for(i=0;i<f1[count].filesize && ptr!=NULL;i++)
{
temp=addblock(ptr);
f1[count].endblock=temp->blockno;
prev=ptr;
freelist=ptr->next;
ptr=ptr->next;
if(allocate==NULL)
{
allocate=temp;
lasta=temp;
}
else
{
lasta->next=temp;
lasta=temp;
}
}
return(allocate);
}

{
```

```
printf("\nEnter the file name:");
scanf("%s",&f1[count].filename);
printf("\nEnter file size in blocks:");
scanf("%d",&f1[count].filesize);
}

void displayfile()
{
int i;
printf("\nFile name\t\tFile size\t\tstart block\t\tEnd block");
for(i=1;i<=count;i++)
{
printf("\n%s",f1[i].filename);
printf("\t\t\t%d",f1[i].filesize);
printf("\t\t\t%d",f1[i].startblock);
printf("\t\t\t%d",f1[i].endblock);
}
}
int main()
{
int ch,result;
char fname[20];
printf("\nEnter the size of disk in blocks");
scanf("%d",&d);
freelist=create();
while(1)
{
printf("\n1: Allocate space for newly created file.");
printf("\n2: Show used and free space on disk.");
printf("\n3: Exit");
printf("\nEnter the choice");
scanf("%d",&ch);
switch(ch)
{
case 1:
acceptfile();
countfree(freelist);
if(countfblock>=f1[count].filesize)
{
allocate=createalloclist();
displayfile();
count++;
}
else
printf("\nNo sufficient space to allocate");
break;
case 2:
printf("\nFree list:");
displaylist(freelist);
printf("\nAllocated list: ");
displaylist(allocate);
break;
case 3:
exit(0);
}
}
}
```

Q.2 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 request 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

Ans →

```
#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);
        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;
}
```