OS Practical solved slip

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:-

```
#include<stdio.h>
#include<unistd.h>
int main()
{
int i,j,k,res,pno;
printf("Enter the no of resourses:");
scanf("%d",&res);
printf("Enter the no of processes :");
scanf("%d",&pno);
int available[res];
printf("Enter the available matrix ");
for(i=0;i<res;i++)
{
scanf("%d",&available[i]);
}
printf("The available matrix is\n ");
for(i=0;i<res;i++)
{
printf("\t r[%d]",i);
}
printf("\n");
for(i=0;i<res;i++)
{
printf("\t %d",available[i]);
}
printf("\nEnter the allocated matrix :");
 int allocation[10][10];
for(i=0;i<pno;i++)
{
for(j=0;j<res;j++)
{
scanf(" %d",&allocation[i][j]);
}
}
printf("The allocation matrix is :\n");
for(i=0;i<pno;i++)
{
for(j=0;j<res;j++)
{
printf(" \t %d ",allocation[i][j]);
}
printf("\n");
}
printf("\nEnter the maximum matrix :"); int Max[10][10];
```

```
for(i=0;i<pno;i++)
{
for(j=0;j<res;j++)
{
scanf(" %d",&Max[i][j]);
}
}
printf("The max matrix is :\n"); for(i=0;i<pno;i++)
{
for(j=0;j<res;j++)
{
printf("\t %d",Max[i][j]);
}
printf("\n");
}
int need[10][10];
for(i=0;i<pno;i++)
{
for(j=0;j<res;j++)
{
need[i][j]=Max[i][j]-allocation[i][j];
}
}
printf("The need matrix is :\n"); for(i=0;i<pno;i++)
{
for(j=0;j<res;j++)
{
printf("\t %d",need[i][j]);
}
printf("\n");
}
}
```

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:-

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int RQ[100],i,n,TotalHeadMovement=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);
for(i=0;i<n;i++)
{
TotalHeadMovement=TotalHeadMovement+abs(RQ[i]-initial);
initial=RQ[i];
}
printf("Total head Movement is %d\n",TotalHeadMovement);
return 0;
}
```

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>
#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<fb;i++)
bit[i] = 0;
printf("\n");
}
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\t\t\tNumber of blocks\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,k,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;
}
```

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

Ans:-

```
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>
int main (int argc, char* argv[]){
int i,my_id, num_procs,N=50;
int array[N],array_final_sum[N],array_final_mult[N];
int r_num,max_value;
unsigned seed;
double t0, t1, time;
```

```
MPI_Init(&argc, &argv);
/* starts MPI */
MPI_Comm_rank (MPI_COMM_WORLD, &my_id);
/* get current process id */
 MPI_Comm_size (MPI_COMM_WORLD, &num_procs);
/* get number of processes */
/*initialization */
t0 = MPI_Wtime();
for(i=0;i<N;i++){
array[i]=my_id +1 ;
}
/* sum and multiplication */
MPI_Reduce(array,array_final_sum,N,MPI_INT,MPI_SUM,0,MPI_COMM_WO
R LD);
MPI_Reduce(array,array_final_mult,N,MPI_INT,MPI_PROD,0,MPI_COMM_W
O RLD);
if(my_id == 0) {
for(i=0;i<N;i++) printf("Final array after sum: %d\n", array_final_sum[i]);
}
if(my_id == 0) {
for(i=0;i<N;i++) printf("Final array after product: %d\n",
array_final_mult[i]) ;
}
/* random number generation */
 seed=my_id+1;
srand(seed);
r_num=rand();
printf("my id is %d and r_num is %d\n", my_id,r_num);
/* calculus of maximum */
MPI_Reduce(&r_num,&max_value,1,MPI_INT,MPI_MAX,0,MPI_COMM_WO
RL D);
/*sleep(10); */
/* time calculation */
t1 = MPI_Wtime();
time = t1 - t0 ;
if(my_id == 0) {
printf("Max_value is (AND THE WINNER IS ....): %d\n", max_value) ;
printf("Total elapsed time [sec] : %f\n", time); }
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>
```

```
#define N 5
#define M 4
int main()
{
 int allocation[N][M] = {{0, 0, 1, 2},
 {1, 0, 0, 0},
 {1, 3, 5, 4},
 {0, 6, 3, 2},
 {0, 0, 1, 4}};
 int max[N][M] = {{0, 0, 1, 2},
 {1, 7, 5, 0},
 {2, 3, 5, 6},
 {0, 6, 5, 2},
 {0, 6, 5, 6}};
 int available[M] = {1, 5, 2, 0};
 int need[N][M];
 // Calculate need matrix
 for (int i = 0; i < N; i++) {
 for (int j = 0; j < M; j++) {
 need[i][j] = max[i][j] - allocation[i][j];
 }
 }
 // Initialize finish array
 int finish[N] = {0};
 // Initialize safe sequence array
 int safe_seq[N];
 // Initialize work array
 int work[M];
 for (int i = 0; i < M; i++) {
 work[i] = available[i];
 }
 // Find safe sequence
 int count = 0;
 while (count < N) {
 int found = 0;
 for (int i = 0; i < N; i++) {
 if (finish[i] == 0) {
 int j;
 for (j = 0; j < M; j++) {
 if (need[i][j] > work[j])
 break;
 }
 if (j == M) {
 for (int k = 0; k < M; k++) {
 work[k] += allocation[i][k];
 }
 safe_seq[count++] = i;
 finish[i] = 1;
 found = 1;
 }
 }
 }
 if (found == 0) {
 printf("System is not in safe state.\n");
 return 0;
 }
 }
 // Print need matrix
 printf("Need matrix:\n");
 for (int i = 0; i < N; i++) {
 printf("P%d: ", i);
 for (int j = 0; j < M; j++) {
 printf("%d ", need[i][j]);
 }
```

```
 printf("\n");
 }
 // Print safe sequence
 printf("System is in safe state.\nSafe sequence is: ");
 for (int i = 0; i < N; i++) {
 printf("P%d ", safe_seq[i]);
 }
 printf("\n");
 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 <stdlib.h>
#include <stdio.h>
#include <mpi.h>
int main (int argc, char* argv[]){
int i,my_id, num_procs,N=50;
int array[N],array_final_sum[N],array_final_mult[N];
int r_num,max_value;
unsigned seed;
double t0, t1, time;
MPI_Init(&argc, &argv);
/* starts MPI */
MPI_Comm_rank (MPI_COMM_WORLD, &my_id);
/* get current process id */
 MPI_Comm_size (MPI_COMM_WORLD, &num_procs);
/* get number of processes */
/*initialization */
t0 = MPI_Wtime();
for(i=0;i<N;i++){
array[i]=my_id +1 ;
}
/* sum and multiplication */
MPI_Reduce(array,array_final_sum,N,MPI_INT,MPI_SUM,0,MPI_COMM_WO
R LD);
MPI_Reduce(array,array_final_mult,N,MPI_INT,MPI_PROD,0,MPI_COMM_W
O RLD);
if(my_id == 0) {
for(i=0;i<N;i++) printf("Final array after sum: %d\n", array_final_sum[i]);
}
if(my_id == 0) {
for(i=0;i<N;i++) printf("Final array after product: %d\n",
array_final_mult[i]) ;
}
/* random number generation */
 seed=my_id+1;
srand(seed);
r_num=rand();
printf("my id is %d and r_num is %d\n", my_id,r_num);
/* calculus of maximum */
MPI_Reduce(&r_num,&max_value,1,MPI_INT,MPI_MAX,0,MPI_COMM_WO
RL D);
/*sleep(10); */
/* time calculation */
t1 = MPI_Wtime();
time = t1 - t0 ;
if(my_id == 0) {
printf("Max_value is (AND THE WINNER IS ....): %d\n", max_value) ;
printf("Total elapsed time [sec] : %f\n", time); }
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>
#include<unistd.h>
int main()
{
int i,j,process,r,p,finish[10],count=0,safeSequence[10];
printf("Enter the no of processes :");
scanf("%d",&p);
for(i=0;i<p;i++)
 {
finish[i]=0;
}
printf("Enter the no of resourses:");
scanf("%d",&r);
int available[r];
printf("Enter the available matrix ");
for(i=0;i<r;i++)
{
scanf("%d",&available[i]);
}
printf("\nEnter the allocated matrix :"); int allocation[10][10];
for(i=0;i<p;i++)
{
printf("\nfor the process %d :",i+1);
 for(j=0;j<r;j++)
{
scanf(" %d",&allocation[i][j]);
}
}
printf("\nEnter the maximum matrix :"); int Max[10][10];
for(i=0;i<p;i++)
{
printf("\nfor the process %d:\n",i+1);
for(j=0;j<r;j++)
{
scanf(" %d",&Max[i][j]);
}
}
 printf("\nThe available matrix is\n ");
for(i=0;i<r;i++)
{
```

```
printf("\t r[%d]",i);
}
printf("\n");
for(i=0;i<r;i++)
{
printf("\t %d",available[i]);
}
printf("\nThe allocation matrix is :\n");
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
{
printf(" \t %d ",allocation[i][j]);
}
 printf("\n");
}
printf("\nThe max matrix is :\n");
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
{
printf("\t %d",Max[i][j]);
}
printf("\n");
}
int need[10][10];
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
{
 need[i][j]=Max[i][j]-allocation[i][j];
}
}
printf("The need matrix is :\n");
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
{
printf("\t %d",need[i][j]);
}
printf("\n");
}
do{
int process=-1;
for(i=0;i<p;i++)
{
if(finish[i]==0)
 {
process=i;
for(j=0;j<r;j++)
{
if(available[j]<need[i][j])
{
process=-1;
break;
}
}
if(process!=-1)
{
printf("\nprocess p%d granted required resources ",process);
safeSequence[count]=process; count++;
for(j=0;j<r;j++)
{
 available[j]+=allocation[process][j];
```

```
allocation[process][j]=0;
Max[process][j]=0;
finish[process]=1;
}
}
}
}
}
while(count!=p && process!=-1);
if(count==p)
{
printf("\nThe system is in safe state :\n"); printf("safe sequence :<");
for(i=0;i<p;i+
+)
printf("p%d ",safeSequence[i]);
printf(">\n");
}
else{
 printf("\n The system is in an unsafe state :");
}
int req[10],proc;
printf("Enter the requested process :\n");
scanf("%d",&proc);
printf("Enter the request\n");
for( i=0;i<r;i++)
{
scanf("%d ",&req[i]);
}
if(req[i]>available[i])
{
printf("The request is not granted immediately \n");
}
else
{
printf("The request is granted immediately\n ");
 }
}
```

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<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);
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(move==1)
{
for(i=index;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];
}
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];
}
}
 else
{
for(i=index-1;i>=0;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];
}
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;
}
```

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<unistd.h>
int main()
{
int i,j,process,r,p,finish[10],count=0,safeSequence[10];
printf("Enter the no of processes :");
scanf("%d",&p);
for(i=0;i<p;i++)
 {
finish[i]=0;
}
printf("Enter the no of resourses:");
scanf("%d",&r);
int available[r];
printf("Enter the available matrix ");
for(i=0;i<r;i++)
{
scanf("%d",&available[i]);
}
printf("\nEnter the allocated matrix :");
int allocation[10][10];
for(i=0;i<p;i++)
{
printf("\nfor the process %d :",i+1);
 for(j=0;j<r;j++)
{
scanf(" %d",&allocation[i][j]);
}
}
printf("\nEnter the maximum matrix :");
 int Max[10][10];
for(i=0;i<p;i++)
{
printf("\nfor the process %d:\n",i+1);
for(j=0;j<r;j++)
{
scanf(" %d",&Max[i][j]);
}
}
 printf("\nThe available matrix is\n ");
for(i=0;i<r;i++)
{
printf("\t r[%d]",i);
}
printf("\n");
for(i=0;i<r;i++)
{
printf("\t %d",available[i]);
}
printf("\nThe allocation matrix is :\n");
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
{
printf(" \t %d ",allocation[i][j]);
}
 printf("\n");
}
printf("\nThe max matrix is :\n");
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
{
```

```
printf("\t %d",Max[i][j]);
}
printf("\n");
}
int need[10][10];
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
{
 need[i][j]=Max[i][j]-allocation[i][j];
}
}
printf("The need matrix is :\n");
for(i=0;i<p;i++)
{
for(j=0;j<r;j++)
{
printf("\t %d",need[i][j]);
}
printf("\n");
}
do{
int process=-1;
for(i=0;i<p;i++)
{
if(finish[i]==0)
 {
process=i;
for(j=0;j<r;j++)
{
if(available[j]<need[i][j])
{
process=-1;
break;
}
}
if(process!=-1)
{
printf("\nprocess p%d granted required resources ",process);
safeSequence[count]=process; count++;
for(j=0;j<r;j++)
{
 available[j]+=allocation[process][j];
allocation[process][j]=0;
Max[process][j]=0;
finish[process]=1;
}
}
}
}
}
while(count!=p && process!=-1);
if(count==p)
{
printf("\nThe system is in safe state :\n");
printf("safe sequence :<");
for(i=0;i<p;i++)
printf("p%d ",safeSequence[i]);
printf(">\n");
}
else{
 printf("\n The system is in an unsafe state :");
}
int req[10],proc;
```

```
printf("Enter the requested process :\n");
scanf("%d",&proc);
printf("Enter the request\n");
for( i=0;i<r;i++)
{
scanf("%d ",&req[i]);
}
if(req[i]>available[i])
{
printf("The request is not granted immediately \n");
}
else
{
printf("The request is granted immediately\n ");
 }
}
```

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 <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define ARRAY_SIZE 1000
int main(int argc, char** argv) {
 int my_rank, num_processes;
 int* array;
 int max_num, local_max;
 int i;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
 MPI_Comm_size(MPI_COMM_WORLD, &num_processes);
 srand(time(NULL) + my_rank);
 array = (int*) malloc(ARRAY_SIZE / num_processes * sizeof(int));
 for (i = 0; i < ARRAY_SIZE / num_processes; i++) {
 array[i] = rand() % 1000;
 }
 local_max = array[0];
 for (i = 1; i < ARRAY_SIZE / num_processes; i++) {
 if (array[i] > local_max) {
 local_max = array[i];
 }
 }
 MPI_Reduce(&local_max, &max_num, 1, MPI_INT, MPI_MAX, 0,
MPI_COMM_WORLD);
 if (my_rank == 0) {
 printf("The maximum number is %d\n", max_num);
 }
 free(array);
 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

```
#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<fb;i++)
bit[i] = 0;
printf("\n");
}
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\t\t\tNumber of blocks\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,k,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;
}
```

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

Answer

```
#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);
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(size1-0);
 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
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

```
#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);
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(move==1)
{
for(i=index;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];
}
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];
}
```

```
}
else
{
for(i=index-1;i>=0;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];
}
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;
}
```

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

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX_BLOCKS 100
int disk[MAX_BLOCKS];
int num_blocks;
int free_blocks;
int *directory;
void init_disk() {
 for (int i = 0; i < num_blocks; i++) {
 disk[i] = rand() % 2; // randomly mark blocks as allocated or free
 if (disk[i] == 0) {
 free_blocks++;
 }
 }
}
void init_directory() {
 directory = (int *)malloc(num_blocks * sizeof(int));
 for (int i = 0; i < num_blocks; i++) {
 directory[i] = -1; // initialize directory with -1
 }
}
void show_bit_vector() {
 printf("Bit Vector:\n");
 for (int i = 0; i < num_blocks; i++) {
 printf("%d ", disk[i]);
 }
 printf("\n");
}
void show_directory() {
```

```
 printf("Directory:\n");
 for (int i = 0; i < num_blocks; i++) {
 printf("%d ", directory[i]);
 }
 printf("\n");
}
void delete_file(int start_block) {
 int current_block = start_block;
 while (current_block != -1) {
 disk[current_block] = 0;
 free_blocks++;
 current_block = directory[current_block];
 directory[current_block] = -1;
 }
 printf("File deleted successfully.\n");
}
int main() {
 printf("Enter the number of blocks in the disk: ");
 scanf("%d", &num_blocks);
 init_disk();
 init_directory();
 int choice, start_block;
 bool running = true;
 while (running) {
 printf("\n1. Show Bit Vector\n2. Show Directory\n3. Delete File\n4. Exit\n");
 printf("Enter your choice: ");
 scanf("%d", &choice);
 switch (choice) {
 case 1:
 show_bit_vector();
 break;
 case 2:
 show_directory();
 break;
 case 3:
 printf("Enter the starting block of the file: ");
 scanf("%d", &start_block);
 delete_file(start_block);
 break;
 case 4:
 running = false;
 break;
 default:
 printf("Invalid choice. Try again.\n");
 }
 }
 return 0;
}
```

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
#include<stdio.h>
#include<stdlib.h>
int main()
{
int RQ[100],i,n,TotalHeadMoment=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);
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;
}
}
TotalHeadMoment=TotalHeadMoment+min;
initial=RQ[index];
RQ[index]=1000;
count++;
}
printf("Total head movement is %d",TotalHeadMoment); 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 safestate or not

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

```
#include <stdio.h>
#include <stdlib.h>
int main() {
 int n, head, i, j, total_head_movements = 0;
 char direction;
 printf("Enter the total number of disk blocks: ");
 scanf("%d", &n);

 int requests[n];

 printf("Enter the disk request string: ");
 for(i=0; i<n; i++) {
 scanf("%d", &requests[i]);
 }
```

```
printf("Enter the current head position: ");
scanf("%d", &head);

printf("Enter the direction (L/R): ");
scanf(" %c", &direction);

// Sort the requests in ascending order
for(i=0; i<n-1; i++) {
for(j=0; j<n-i-1; j++) {
if(requests[j] > requests[j+1]) {
int temp = requests[j];
requests[j] = requests[j+1];
requests[j+1] = temp;
}
}
}

printf("\nOrder of disk request served: ");

if(direction == 'L') {
// Scan left from the head
for(i=0; i<n; i++) {
if(requests[i] >= head) {
break; // Stop at first request greater than or equal to head
}
}

for(j=i; j>=0; j--) {
printf("%d ", requests[j]);
total_head_movements += abs(head - requests[j]);

head = requests[j];
}

for(j=i+1; j<n; j++) {
printf("%d ", requests[j]);
total_head_movements += abs(head - requests[j]);
head = requests[j];
}
}
else {
// Scan right from the head
for(i=n-1; i>=0; i--) {
if(requests[i] <= head) {
break; // Stop at first request less than or equal to head
}
}

for(j=i; j<n; j++) {
printf("%d ", requests[j]);
total_head_movements += abs(head - requests[j]);
head = requests[j];
}

for(j=i-1; j>=0; j--) {
printf("%d ", requests[j]);
total_head_movements += abs(head - requests[j]);
head = requests[j];
}
}

printf("\nTotal head movements: %d", total_head_movements);
```

```
 return 0;
}
```

Slip 10

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

```
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>
int main (int argc, char* argv[]){
int i,my_id, num_procs,N=50;
int array[N],array_final_sum[N],array_final_mult[N];
int r_num,max_value;
unsigned seed;
double t0, t1, time;
MPI_Init(&argc, &argv); /* starts MPI */
MPI_Comm_rank (MPI_COMM_WORLD, &my_id); /* get current process id */
 MPI_Comm_size (MPI_COMM_WORLD, &num_procs);
 /* get number of
processes */
/*initialization */
t0 = MPI_Wtime();
for(i=0;i<N;i++){
array[i]=my_id +1 ;
}
/* sum and multiplication */
MPI_Reduce(array,array_final_sum,N,MPI_INT,MPI_SUM,0,MPI_COMM_WO
R LD);
MPI_Reduce(array,array_final_mult,N,MPI_INT,MPI_PROD,0,MPI_COMM_W
O RLD);
if(my_id == 0) {
for(i=0;i<N;i++) printf("Final array after sum: %d\n", array_final_sum[i]);
}
if(my_id == 0) {
for(i=0;i<N;i++) printf("Final array after product: %d\n",
array_final_mult[i]) ;
}
/* random number generation */
 seed=my_id+1;
srand(seed);
r_num=rand();
printf("my id is %d and r_num is %d\n", my_id,r_num);
/* calculus of maximum */
MPI_Reduce(&r_num,&max_value,1,MPI_INT,MPI_MAX,0,MPI_COMM_WO
RL D);
/*sleep(10); */
/* time calculation */
t1 = MPI_Wtime();
time = t1 - t0 ;
if(my_id == 0) {
printf("Max_value is (AND THE WINNER IS ....): %d\n", max_value) ;
printf("Total elapsed time [sec] : %f\n", time); }
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
#include<stdio.h>
#include<stdlib.h>
int main(){
 int num_blocks, head_pos, total_head_movements=0, current_pos, i, j, temp;
 char direction;
 printf("Enter the number of disk blocks: ");
 scanf("%d", &num_blocks);
 int requests[num_blocks];
 printf("Enter the disk request string: ");
 for(i=0; i<num_blocks; i++){
 scanf("%d", &requests[i]);
 }
 printf("Enter the starting head position: ");
 scanf("%d", &head_pos);
 printf("Enter the direction (L/R): ");
 scanf(" %c", &direction);
 // Sorting requests in ascending order
 for(i=0; i<num_blocks-1; i++){
 for(j=0; j<num_blocks-i-1; j++){
 if(requests[j] > requests[j+1]){
 temp = requests[j];
 requests[j] = requests[j+1];
 requests[j+1] = temp;
 }
 }
 }

 printf("\nOrder of served requests: ");
 // C-SCAN algorithm implementation
 if(direction == 'R'){
 for(i=0; i<num_blocks; i++){
 if(requests[i] >= head_pos){
 printf("%d ", requests[i]);
 total_head_movements += requests[i] - head_pos;
 head_pos = requests[i];
 }
 }
 printf("%d ", 199);
 total_head_movements += 199 - head_pos;
 head_pos = 0;
 for(i=0; i<num_blocks; i++){
 if(requests[i] < head_pos){
 printf("%d ", requests[i]);
 total_head_movements += 199 - head_pos + requests[i];
 head_pos = requests[i];
 }
 }
 }
 else if(direction == 'L'){
 for(i=num_blocks-1; i>=0; i--){
 if(requests[i] <= head_pos){
 printf("%d ", requests[i]);
 total_head_movements += head_pos - requests[i];
 head_pos = requests[i];
 }
 }
```

```
 printf("%d ", 0);
 total_head_movements += head_pos;
 head_pos = 199;
 for(i=num_blocks-1; i>=0; i--){
 if(requests[i] > head_pos){
 printf("%d ", requests[i]);
 total_head_movements += 199 - requests[i] + head_pos;
 head_pos = requests[i];
 }
 }
 }
 printf("\nTotal number of head movements: %d\n", total_head_movements);
 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

```
#include <stdio.h>
#define N 5
#define M 4
int main()
{
 int allocation[N][M] = {{0, 0, 1, 2}, //P0
 {1, 0, 0, 0}, //P1
 {1, 3, 5, 4}, //P2
 {0, 6, 3, 2}, //P3
 {0, 0, 1, 4}};//P4
 int max[N][M] = {{0, 0, 1, 2},
 {1, 7, 5, 0},
 {2, 3, 5, 6},
 {0, 6, 5, 2},
{0, 6, 5, 6}};
 int available[M] = {1, 5, 2, 0};
int need[N][M];
 // Calculate need matrix
 for (int i = 0; i < N; i++) {
 for (int j = 0; j < M; j++) {
 need[i][j] = max[i][j] - allocation[i][j];
 }
 }
// Initialize finish array
 int finish[N] = {0};
```

```
// Initialize safe sequence array
 int safe_seq[N];
// Initialize work array
 int work[M];
 for (int i = 0; i < M; i++) {

 work[i] = available[i];
 }
// Find safe sequence
 int count = 0;
 while (count < N) {
 int found = 0;
 for (int i = 0; i < N; i++) {
 if (finish[i] == 0) {
 int j;
 for (j = 0; j < M; j++) {
 if (need[i][j] > work[j])
 break;
 }
 if (j == M) {
 for (int k = 0; k < M; k++) {
 work[k] += allocation[i][k];
 }
 safe_seq[count++] = i;
 finish[i] = 1;
 found = 1;
 }
 }
 }
 if (found == 0) {
 printf("System is not in safe state.\n");
 return 0;
 }
 }
// Print need matrix
 printf("Need matrix:\n");
 for (int i = 0; i < N; i++) {
 printf("P%d: ", i);
 for (int j = 0; j < M; j++) {

 printf("%d ", need[i][j]);
 }
 printf("\n");
 }
 // Print safe sequence
 printf("System is in safe state.\nSafe sequence is: ");
 for (int i = 0; i < N; i++) {
 printf("P%d ", safe_seq[i]);
 }
 printf("\n");
 return 0;
}
```

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)

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>
#define ARRAY_SIZE 1000
int main(int argc, char** argv) {
 int my_rank, num_processes;
 int* array;
 int local_min = INT_MAX, global_min;
```

```
 int i;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
 MPI_Comm_size(MPI_COMM_WORLD, &num_processes);
 srand(time(NULL) + my_rank);
 array = (int*) malloc(ARRAY_SIZE / num_processes * sizeof(int));
 for (i = 0; i < ARRAY_SIZE / num_processes; i++) {
 array[i] = rand() % 1000;
 if (array[i] < local_min) {
 local_min = array[i];
 }
 }
 MPI_Reduce(&local_min, &global_min, 1, MPI_INT, MPI_MIN, 0,
MPI_COMM_WORLD);
 if (my_rank == 0) {
 printf("The minimum number is %d\n", global_min);
 }
 free(array);
 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

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

```
#include <stdio.h>
#include <stdlib.h>
int main() {
 int n, head, i, j, total_head_movements = 0;
 char direction;

 printf("Enter the total number of disk blocks: ");
 scanf("%d", &n);

 int requests[n];

 printf("Enter the disk request string: ");
 for(i=0; i<n; i++) {
 scanf("%d", &requests[i]);
 }

 printf("Enter the current head position: ");
 scanf("%d", &head);

 printf("Enter the direction (L/R): ");
 scanf(" %c", &direction);

 // Sort the requests in ascending order
```

```
 for(i=0; i<n-1; i++) {
 for(j=0; j<n-i-1; j++) {
 if(requests[j] > requests[j+1]) {
 int temp = requests[j];
 requests[j] = requests[j+1];
 requests[j+1] = temp;
 }
 }
 }


 printf("\nOrder of disk request served: ");
 if(direction == 'L') {
 // Scan left from the head
 for(i=n-1; i>=0; i--) {
 if(requests[i] <= head) {
 break; // Stop at first request less than or equal to head
 }
 }

 for(j=i; j>=0; j--) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }

 for(j=n-1; j>i; j--) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }
 }
 else {
 // Scan right from the head
 for(i=0; i<n; i++) {
 if(requests[i] >= head) {
 break; // Stop at first request greater than or equal to head
 }
 }

 for(j=i; j<n; j++) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }

 for(j=0; j<i; j++) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }
 }

 printf("\nTotal head movements: %d", total_head_movements);

 return 0;
}
Output:Enter the total number of disk blocks: 8
Enter the disk request string: 23
89
132
42
187
69
```

```
36
55
Enter the current head position: 40
Enter the direction (L/R): R
Order of disk request served: 42 55 69 89 132 187 23 36
Total head movements: 324
```

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

```
#include <stdio.h>
#define N 5
#define M 4
int main()
{
 int allocation[N][M] = {{0, 0, 1, 2}, //P0
 {1, 0, 0, 0}, //P1
 {1, 3, 5, 4}, //P2
 {0, 6, 3, 2}, //P3
 {0, 0, 1, 4}};//P4

 int max[N][M] = {{0, 0, 1, 2},
 {1, 7, 5, 0},
 {2, 3, 5, 6},
 {0, 6, 5, 2},
 {0, 6, 5, 6}};
 int available[M] = {1, 5, 2, 0};
 int need[N][M];
 // Calculate need matrix
 for (int i = 0; i < N; i++) {
 for (int j = 0; j < M; j++) {
 need[i][j] = max[i][j] - allocation[i][j];
 }
 }
 // Initialize finish array
 int finish[N] = {0};
 // Initialize safe sequence array
 int safe_seq[N];
 // Initialize work array
 int work[M];
 for (int i = 0; i < M; i++) {
 work[i] = available[i];
 }
 // Find safe sequence
 int count = 0;
 while (count < N) {
```

```
 int found = 0;
 for (int i = 0; i < N; i++) {
 if (finish[i] == 0) {
 int j;
 for (j = 0; j < M; j++) {
 if (need[i][j] > work[j])

 break;
 }
 if (j == M) {
 for (int k = 0; k < M; k++) {
 work[k] += allocation[i][k];
 }
 safe_seq[count++] = i;
 finish[i] = 1;
 found = 1;
 }
 }
 }
 if (found == 0) {
 printf("System is not in safe state.\n");
 return 0;
 }
 }
 // Print need matrix
 printf("Need matrix:\n");
 for (int i = 0; i < N; i++) {
 printf("P%d: ", i);
 for (int j = 0; j < M; j++) {
 printf("%d ", need[i][j]);
 }
 printf("\n");
 }
 // Print safe sequence
 printf("System is in safe state.\nSafe sequence is: ");
 for (int i = 0; i < N; i++) {
 printf("P%d ", safe_seq[i]);
 }

 printf("\n");
 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

```
#include<stdio.h>
#include<stdlib.h>
int main(){
 int num_blocks, head_pos, total_head_movements=0, current_pos, i, j, temp;
 char direction;
 printf("Enter the number of disk blocks: ");
 scanf("%d", &num_blocks);
 int requests[num_blocks];
 printf("Enter the disk request string: ");
 for(i=0; i<num_blocks; i++){
 scanf("%d", &requests[i]);
 }
 printf("Enter the starting head position: ");
```

```
 scanf("%d", &head_pos);
 printf("Enter the direction (L/R): ");
 scanf(" %c", &direction);
 // Sorting requests in ascending order
 for(i=0; i<num_blocks-1; i++){
 for(j=0; j<num_blocks-i-1; j++){
 if(requests[j] > requests[j+1]){
 temp = requests[j];
 requests[j] = requests[j+1];
 requests[j+1] = temp;
 }
 }
 }
 printf("\nOrder of served requests: ");
 // SCAN algorithm implementation
 if(direction == 'L'){
 for(i=0; i<num_blocks; i++){
 if(requests[i] < head_pos){
 printf("%d ", requests[i]);
 total_head_movements += head_pos - requests[i];
 head_pos = requests[i];
 }
 }
 printf("%d ", 0);
 total_head_movements += head_pos;
 head_pos = 0;
 for(i=num_blocks-1; i>=0; i--){
 if(requests[i] >= head_pos){
 printf("%d ", requests[i]);
 total_head_movements += requests[i] - head_pos;
 head_pos = requests[i];
 }
 }
 }
 else if(direction == 'R'){
 for(i=num_blocks-1; i>=0; i--){
 if(requests[i] > head_pos){
 printf("%d ", requests[i]);
 total_head_movements += requests[i] - head_pos;
 head_pos = requests[i];
 }
 }
 printf("%d ", 199);
 total_head_movements += 199 - head_pos;
 head_pos = 199;
 for(i=0; i<num_blocks; i++){
 if(requests[i] <= head_pos){
 printf("%d ", requests[i]);
 total_head_movements += head_pos - requests[i];
 head_pos = requests[i];
 }
 }
 }
 printf("\nTotal number of head movements: %d\n", total_head_movements);
 return 0;
}
Output:Enter the number of disk blocks: 8
Enter the disk request string: 176
79
34
60
92
11
41
```

```
114
Enter the starting head position: 65
Enter the direction (L/R): L
Order of served requests: 11 0 176
Total number of head movements: 241
```

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

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX_BLOCKS 100
int disk[MAX_BLOCKS];
int num_blocks;
int free_blocks;
int *directory;
void init_disk() {
 for (int i = 0; i < num_blocks; i++) {
 disk[i] = rand() % 2; // randomly mark blocks as allocated or free
 if (disk[i] == 0) {
 free_blocks++;
 }
 }
}
void init_directory() {
 directory = (int *)malloc(num_blocks * sizeof(int));
 for (int i = 0; i < num_blocks; i++) {
 directory[i] = -1; // initialize directory with -1
 }
}
void show_bit_vector() {
 printf("Bit Vector:\n");
 for (int i = 0; i < num_blocks; i++) {
 printf("%d ", disk[i]);
 }
 printf("\n");
}
void show_directory() {
 printf("Directory:\n");
 for (int i = 0; i < num_blocks; i++) {
 printf("%d ", directory[i]);
 }
 printf("\n");
}
void delete_file(int start_block) {
 int current_block = start_block;
 while (current_block != -1) {
 disk[current_block] = 0;
 free_blocks++;
 current_block = directory[current_block];
```

```
 directory[current_block] = -1;
 }
 printf("File deleted successfully.\n");
}
int main() {
 printf("Enter the number of blocks in the disk: ");
 scanf("%d", &num_blocks);
 init_disk();
 init_directory();
 int choice, start_block;
 bool running = true;
 while (running) {
 printf("\n1. Show Bit Vector\n2. Show Directory\n3. Delete File\n4. Exit\n");
 printf("Enter your choice: ");
 scanf("%d", &choice);
 switch (choice) {
 case 1:
 show_bit_vector();
 break;
 case 2:
 show_directory();
 break;
 case 3:
 printf("Enter the starting block of the file: ");
 scanf("%d", &start_block);
 delete_file(start_block);
 break;
 case 4:
 running = false;
 break;
 default:
 printf("Invalid choice. Try again.\n");
 }
 }
 return 0;
}
```

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

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main()
{
 int n, head, i, j, min, min_index, total_head_movements = 0;
 printf("Enter the total number of disk blocks: ");
 scanf("%d", &n);
 int requests[n], completed[n];

 printf("Enter the disk request string: ");
 for(i=0; i<n; i++) {
 scanf("%d", &requests[i]);
 completed[i] = 0; // Initialize all requests as incomplete
 }

 printf("Enter the current head position: ");
 scanf("%d", &head);
 printf("\nOrder of disk request served: ");
```

```
 for(i=0; i<n; i++) {
 min = __INT_MAX__;
 for(j=0; j<n; j++) {
 if(completed[j] == 1) {
 continue; // Skip completed requests
 }

 if(abs(head - requests[j]) < min) {
 min = abs(head - requests[j]);
 min_index = j;
 }
 }

 completed[min_index] = 1; // Mark request as completed
 total_head_movements += min; // Add distance to total head movements
 head = requests[min_index]; // Move head to the selected request
 printf("%d ", requests[min_index]);
 }

 printf("\nTotal head movements: %d", total_head_movements);
 return 0;
}
```

```
Output:Enter the total number of disk blocks: 9
Enter the disk request string: 55
58
39
18
90
160
150
38
184
Enter the current head position: 50
Order of disk request served: 55 58 39 38 18 90 150 160 184
Total head movements: 214
```

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

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_BLOCKS 100
// Function to display the bit vector
void showBitVector(int* blocks, int n) {
 printf("Bit Vector:\n");
 for (int i = 0; i < n; i++) {
 printf("%d ", blocks[i]);
 }
 printf("\n");
}
// Function to create a new file
```

```
void createFile(int* blocks, int* head, int* tail, int n) {
 int start, size;
 printf("Enter the starting block: ");
 scanf("%d", &start);
 printf("Enter the size of file: ");
 scanf("%d", &size);
 if (start < 0 || start >= n || blocks[start] != 0) {
 printf("Invalid starting block\n");
 return;
 }
 int count = 1;
 int current = start;
 while (count < size) {
 if (current == -1) {
 printf("Not enough space available\n");
 return;
 }
 current = tail[current];
 count++;
 }
 int newBlock = -1;
 for (int i = 0; i < n; i++) {
 if (blocks[i] == 0) {
 newBlock = i;
 blocks[newBlock] = -1;
 break;
 }
 }
 if (newBlock == -1) {
 printf("Not enough space available\n");
 return;
 }
 tail[current] = newBlock;
 head[newBlock] = current;
 tail[newBlock] = -1;
 blocks[newBlock] = 1;
 printf("File created successfully\n");
}
// Function to display the directory
void showDirectory(int* head, int* tail, int n) {
 printf("Directory:\n");
 for (int i = 0; i < n; i++) {
 if (head[i] != -1) {
 printf("%d ", i);
 int current = head[i];
 while (current != -1) {
 printf("%d ", current);
 current = tail[current];
 }
 printf("\n");
 }
 }
}
int main() {
 int n;
 printf("Enter the number of blocks: ");
 scanf("%d", &n);
 int blocks[MAX_BLOCKS] = {0}; // 0 - free block, -1 - used block
 int head[MAX_BLOCKS], tail[MAX_BLOCKS];
 for (int i = 0; i < n; i++) {
 head[i] = -1;
 tail[i] = -1;
 }
 while (1) {
```

```
 printf("\n1. Show Bit Vector\n");
 printf("2. Create New File\n");
 printf("3. Show Directory\n");
 printf("4. Exit\n");
 int choice;
 printf("Enter your choice: ");
 scanf("%d", &choice);
 switch (choice) {
 case 1:
 showBitVector(blocks, n);
 break;
 case 2:
 createFile(blocks, head, tail, n);
 break;
 case 3:
 showDirectory(head, tail, n);
 break;
 case 4:
 exit(0);
 default:
 printf("Invalid choice\n");
 }
 }
 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.. [15]

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

Starting Head Position: 70

Direction: Right

```
#include<stdio.h>
#include<stdlib.h>
int main(){
 int num_blocks, head_pos, total_head_movements=0, current_pos, i, j, temp;
 char direction;
 printf("Enter the number of disk blocks: ");
 scanf("%d", &num_blocks);
 int requests[num_blocks];
 printf("Enter the disk request string: ");
 for(i=0; i<num_blocks; i++){
 scanf("%d", &requests[i]);
 }
 printf("Enter the starting head position: ");
 scanf("%d", &head_pos);
 printf("Enter the direction (L/R): ");
 scanf(" %c", &direction);
 // Sorting requests in ascending order
 for(i=0; i<num_blocks-1; i++){
 for(j=0; j<num_blocks-i-1; j++){
 if(requests[j] > requests[j+1]){
 temp = requests[j];
 requests[j] = requests[j+1];
 requests[j+1] = temp;
 }
 }
 }
 printf("\nOrder of served requests: ");
 // C-SCAN algorithm implementation

 if(direction == 'R'){
```

```
 for(i=0; i<num_blocks; i++){
 if(requests[i] >= head_pos){
 printf("%d ", requests[i]);
 total_head_movements += requests[i] - head_pos;
 head_pos = requests[i];
 }
 }
 printf("%d ", 199);
 total_head_movements += 199 - head_pos;
 head_pos = 0;
 for(i=0; i<num_blocks; i++){
 if(requests[i] < head_pos){
 printf("%d ", requests[i]);
 total_head_movements += 199 - head_pos + requests[i];
 head_pos = requests[i];
 }
 }
 }
 else if(direction == 'L'){
 for(i=num_blocks-1; i>=0; i--){
 if(requests[i] <= head_pos){
 printf("%d ", requests[i]);
 total_head_movements += head_pos - requests[i];
 head_pos = requests[i];
 }
 }
 printf("%d ", 0);
 total_head_movements += head_pos;
 head_pos = 199;
 for(i=num_blocks-1; i>=0; i--){
 if(requests[i] > head_pos){
 printf("%d ", requests[i]);
 total_head_movements += 199 - requests[i] + head_pos;
 head_pos = requests[i];
 }
 }
 }
 printf("\nTotal number of head movements: %d\n", total_head_movements);
 return 0;
}
Output:Enter the number of disk blocks: 7
Enter the disk request string: 80
150
60
135
40
35
170
Enter the starting head position: 70
Enter the direction (L/R): R
Order of served requests: 80 135 150 170 199
Total number of head movements: 129
```

Slip 17

Q.1 Write a program to simulate Indexed 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
- Show Directory
- Delete Already File
- Exit

-->

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX_BLOCKS 100
typedef struct {
 bool allocated;
 int index;
} Block;
int main() {
 int n;
 printf("Enter the number of blocks on the disk: ");
 scanf("%d", &n);
 // Initialize the bit vector and directory
 Block bit_vector[MAX_BLOCKS];
 int directory[MAX_BLOCKS];
 for (int i = 0; i < n; i++) {
 bit_vector[i].allocated = false;
 bit_vector[i].index = -1;
 directory[i] = -1;
 }
 // Mark some blocks as allocated
 for (int i = 0; i < n / 2; i++) {
 int block_index = rand() % n;
 bit_vector[block_index].allocated = true;
 bit_vector[block_index].index = i;
 directory[i] = block_index;
 }
 int choice;
 do {
 // Display the menu options
 printf("\n1. Show Bit Vector\n");
 printf("2. Show Directory\n");
 printf("3. Delete Already File\n");
 printf("4. Exit\n");
 printf("Enter your choice: ");
 scanf("%d", &choice);
 switch (choice) {
 case 1:
 // Show the bit vector
 printf("Bit Vector:\n");
 for (int i = 0; i < n; i++) {
 printf("%d ", bit_vector[i].allocated);
 }
 printf("\n");
 break;
 case 2:
 // Show the directory
 printf("Directory:\n");
 for (int i = 0; i < n; i++) {
 if (directory[i] != -1) {
 printf("%d -> %d\n", i, directory[i]);
 }
 }
 break;
 case 3:
 // Delete a file
```

```
 int file_index;
 printf("Enter the file index to delete: ");
 scanf("%d", &file_index);
 if (directory[file_index] == -1) {
 printf("File not found.\n");
 } else {
 int block_index = directory[file_index];
 bit_vector[block_index].allocated = false;
 bit_vector[block_index].index = -1;
 directory[file_index] = -1;
 printf("File deleted successfully.\n");
 }
 break;
 case 4:
 // Exit the program
 printf("Exiting...\n");
 break;
 default:
 printf("Invalid choice. Please try again.\n");
 }
 } while (choice != 4);
 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.

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

Start Head Position: 40

Direction: Left

```
#include <stdio.h>
#include <stdlib.h>
int main() {
 int n, head, i, j, total_head_movements = 0;
 char direction;

 printf("Enter the total number of disk blocks: ");
 scanf("%d", &n);

 int requests[n];

 printf("Enter the disk request string: ");
 for(i=0; i<n; i++) {
 scanf("%d", &requests[i]);
 }

 printf("Enter the current head position: ");
 scanf("%d", &head);

 printf("Enter the direction (L/R): ");
 scanf(" %c", &direction);

 // Sort the requests in ascending order
 for(i=0; i<n-1; i++) {
 for(j=0; j<n-i-1; j++) {
 if(requests[j] > requests[j+1]) {
 int temp = requests[j];
 requests[j] = requests[j+1];
 requests[j+1] = temp;
 }
 }
 }
```

```
 printf("\nOrder of disk request served: ");

 if(direction == 'L') {
 // Scan left from the head
 for(i=0; i<n; i++) {
 if(requests[i] >= head) {
 break; // Stop at first request greater than or equal to head
 }
 }

 for(j=i; j>=0; j--) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }

 for(j=i+1; j<n; j++) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }
 }
 else {
 // Scan right from the head
 for(i=n-1; i>=0; i--) {
 if(requests[i] <= head) {
 break; // Stop at first request less than or equal to head
 }
 }

 for(j=i; j<n; j++) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }

 for(j=i-1; j>=0; j--) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }
 }

 printf("\nTotal head movements: %d", total_head_movements);

 return 0;
}
```

Slip 18

Q.1 Write a program to simulate Indexed 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
• Delete File
• Exit

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX_BLOCKS 100
typedef struct {
 bool allocated;
 int index;
} Block;
int main() {
 int n;
 printf("Enter the number of blocks on the disk: ");
 scanf("%d", &n);
 // Initialize the bit vector and directory
 Block bit_vector[MAX_BLOCKS];
 int directory[MAX_BLOCKS];
 for (int i = 0; i < n; i++) {
 bit_vector[i].allocated = false;
 bit_vector[i].index = -1;
 directory[i] = -1;
 }
 // Mark some blocks as allocated
 for (int i = 0; i < n / 2; i++) {
 int block_index = rand() % n;
 bit_vector[block_index].allocated = true;
 bit_vector[block_index].index = i;
 directory[i] = block_index;
 }
 int choice;
 do {
 // Display the menu options
 printf("\n1. Show Bit Vector\n");
 printf("2. Show Directory\n");
 printf("3. Delete Already File\n");
 printf("4. Exit\n");
 printf("Enter your choice: ");
 scanf("%d", &choice);
 switch (choice) {
 case 1:
 // Show the bit vector
 printf("Bit Vector:\n");
 for (int i = 0; i < n; i++) {
 printf("%d ", bit_vector[i].allocated);
 }
 printf("\n");
 break;
 case 2:
 // Show the directory
 printf("Directory:\n");
 for (int i = 0; i < n; i++) {
 if (directory[i] != -1) {
 printf("%d -> %d\n", i, directory[i]);
 }
 }
 break;
 case 3:
 // Delete a file
 int file_index;
 printf("Enter the file index to delete: ");
 scanf("%d", &file_index);
 if (directory[file_index] == -1) {
```

```
 printf("File not found.\n");
 } else {
 int block_index = directory[file_index];
 bit_vector[block_index].allocated = false;
 bit_vector[block_index].index = -1;
 directory[file_index] = -1;
 printf("File deleted successfully.\n");
 }
 break;
 case 4:
 // Exit the program
 printf("Exiting...\n");
 break;
 default:
 printf("Invalid choice. Please try again.\n");
 }
 } while (choice != 4);
 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.

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

Start Head Position: 72

Direction: Right

```
#include<stdio.h>
#include<stdlib.h>
int main(){
 int num_blocks, head_pos, total_head_movements=0, current_pos, i, j, temp;
 char direction;
 printf("Enter the number of disk blocks: ");
 scanf("%d", &num_blocks);
 int requests[num_blocks];
 printf("Enter the disk request string: ");
 for(i=0; i<num_blocks; i++){
 scanf("%d", &requests[i]);
 }
 printf("Enter the starting head position: ");
 scanf("%d", &head_pos);
 printf("Enter the direction (L/R): ");
 scanf(" %c", &direction);
 // Sorting requests in ascending order
 for(i=0; i<num_blocks-1; i++){
 for(j=0; j<num_blocks-i-1; j++){
 if(requests[j] > requests[j+1]){
 temp = requests[j];
 requests[j] = requests[j+1];
 requests[j+1] = temp;
 }
 }
 }
 printf("\nOrder of served requests: ");
 // SCAN algorithm implementation
 if(direction == 'L'){
 for(i=0; i<num_blocks; i++){
 if(requests[i] < head_pos){
 printf("%d ", requests[i]);
 total_head_movements += head_pos - requests[i];
 head_pos = requests[i];
 }
 }
```

```
 printf("%d ", 0);
 total_head_movements += head_pos;
 head_pos = 0;
 for(i=num_blocks-1; i>=0; i--){
 if(requests[i] >= head_pos){
 printf("%d ", requests[i]);
 total_head_movements += requests[i] - head_pos;
 head_pos = requests[i];
 }
 }
 }
 else if(direction == 'R'){
 for(i=num_blocks-1; i>=0; i--){
 if(requests[i] > head_pos){
 printf("%d ", requests[i]);
 total_head_movements += requests[i] - head_pos;
 head_pos = requests[i];
 }
 }
 printf("%d ", 199);
 total_head_movements += 199 - head_pos;
 head_pos = 199;
 for(i=0; i<num_blocks; i++){
 if(requests[i] <= head_pos){
 printf("%d ", requests[i]);
 total_head_movements += head_pos - requests[i];
 head_pos = requests[i];
 }
 }
 }
 printf("\nTotal number of head movements: %d\n", total_head_movements);
 return 0;
}
```

Slip 19

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				A vailable			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	3	2	4	6	5	4	4	3	4	4	2
P1	1	2	0	1	4	4	4	4				
P2	0	0	0	0	0	0	1	2				
P3	3	3	2	2	3	9	3	4				
P4	1	4	3	2	2	5	3	3				
P5	2	4	1	4	4	6	3	4				

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

```
#include <stdio.h>
#define N 5
#define M 4
int main()
{
 int allocation[N][M] = {{0, 0, 1, 2},
 {1, 0, 0, 0},
```

```
{1, 3, 5, 4},
{0, 6, 3, 2},
{0, 0, 1, 4}};
int max[N][M] = {{0, 0, 1, 2},
{1, 7, 5, 0},
{2, 3, 5, 6},
{0, 6, 5, 2},
{0, 6, 5, 6}};
int available[M] = {1, 5, 2, 0};
int need[N][M];
// Calculate need matrix
for (int i = 0; i < N; i++) {
for (int j = 0; j < M; j++) {
need[i][j] = max[i][j] - allocation[i][j];
}
}
// Initialize finish array
int finish[N] = {0};
// Initialize safe sequence array
int safe_seq[N];
// Initialize work array
int work[M];
for (int i = 0; i < M; i++) {
work[i] = available[i];
}
// Find safe sequence
int count = 0;
while (count < N) {
int found = 0;
for (int i = 0; i < N; i++) {
if (finish[i] == 0) {
int j;
for (j = 0; j < M; j++) {
if (need[i][j] > work[j])
break;
}
if (j == M) {
for (int k = 0; k < M; k++) {
work[k] += allocation[i][k];
}
safe_seq[count++] = i;
finish[i] = 1;
found = 1;
}
}
}
if (found == 0) {
printf("System is not in safe state.\n");
return 0;
}
}
// Print need matrix
printf("Need matrix:\n");
for (int i = 0; i < N; i++) {
printf("P%d: ", i);
for (int j = 0; j < M; j++) {
printf("%d ", need[i][j]);
}
printf("\n");
}
// Print safe sequence
printf("System is in safe state.\nSafe sequence is: ");
for (int i = 0; i < N; i++) {
printf("P%d ", safe_seq[i]);
```

```
 }
 printf("\n");
 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.

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

Start Head Position: 40

Direction: Left

```
#include<stdio.h>
#include<stdlib.h>
int main(){
 int num_blocks, head_pos, total_head_movements=0, current_pos, i, j, temp;
 char direction;
 printf("Enter the number of disk blocks: ");
 scanf("%d", &num_blocks);
 int requests[num_blocks];
 printf("Enter the disk request string: ");
 for(i=0; i<num_blocks; i++){
 scanf("%d", &requests[i]);
 }
 printf("Enter the starting head position: ");
 scanf("%d", &head_pos);
 printf("Enter the direction (L/R): ");
 scanf(" %c", &direction);
 // Sorting requests in ascending order
 for(i=0; i<num_blocks-1; i++){
 for(j=0; j<num_blocks-i-1; j++){
 if(requests[j] > requests[j+1]){
 temp = requests[j];
 requests[j] = requests[j+1];
 requests[j+1] = temp;
 }
 }
 }
 printf("\nOrder of served requests: ");
 // C-SCAN algorithm implementation
 if(direction == 'R'){
 for(i=0; i<num_blocks; i++){
 if(requests[i] >= head_pos){
 printf("%d ", requests[i]);
 total_head_movements += requests[i] - head_pos;
 head_pos = requests[i];
 }
 }
 printf("%d ", 199);
 total_head_movements += 199 - head_pos;
 head_pos = 0;
 for(i=0; i<num_blocks; i++){
 if(requests[i] < head_pos){
 printf("%d ", requests[i]);
 total_head_movements += 199 - head_pos + requests[i];
 head_pos = requests[i];
 }
 }
 }
 else if(direction == 'L'){
 for(i=num_blocks-1; i>=0; i--){
 if(requests[i] <= head_pos){
```

```
 printf("%d ", requests[i]);
 total_head_movements += head_pos - requests[i];
 head_pos = requests[i];
 }
 }
 printf("%d ", 0);
 total_head_movements += head_pos;
 head_pos = 199;
 for(i=num_blocks-1; i>=0; i--){
 if(requests[i] > head_pos){
 printf("%d ", requests[i]);
 total_head_movements += 199 - requests[i] + head_pos;
 head_pos = requests[i];
 }
 }
 }
 printf("\nTotal number of head movements: %d\n", total_head_movements);
 return 0;
}
```

slip 20

Q.1 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.

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

Start Head Position: 72

Direction: User defined

-->

```
#include<stdio.h>
#include<stdlib.h>
int main(){
 int num_blocks, head_pos, total_head_movements=0, current_pos, i, j, temp;
 char direction;
 printf("Enter the number of disk blocks: ");
 scanf("%d", &num_blocks);
 int requests[num_blocks];
 printf("Enter the disk request string: ");
 for(i=0; i<num_blocks; i++){
 scanf("%d", &requests[i]);
 }
 printf("Enter the starting head position: ");
 scanf("%d", &head_pos);
 printf("Enter the direction (L/R): ");
 scanf(" %c", &direction);
 // Sorting requests in ascending order
 for(i=0; i<num_blocks-1; i++){
 for(j=0; j<num_blocks-i-1; j++){
 if(requests[j] > requests[j+1]){
 temp = requests[j];
 requests[j] = requests[j+1];
 requests[j+1] = temp;
 }
 }
 }
```

```
 printf("\nOrder of served requests: ");
 // SCAN algorithm implementation
 if(direction == 'L'){
 for(i=0; i<num_blocks; i++){
 if(requests[i] < head_pos){
 printf("%d ", requests[i]);
 total_head_movements += head_pos - requests[i];
 head_pos = requests[i];
 }
 }
 printf("%d ", 0);
 total_head_movements += head_pos;
 head_pos = 0;
 for(i=num_blocks-1; i>=0; i--){
 if(requests[i] >= head_pos){
 printf("%d ", requests[i]);
 total_head_movements += requests[i] - head_pos;
 head_pos = requests[i];
 }
 }
 }
 else if(direction == 'R'){
 for(i=num_blocks-1; i>=0; i--){
 if(requests[i] > head_pos){
 printf("%d ", requests[i]);
 total_head_movements += requests[i] - head_pos;
 head_pos = requests[i];
 }
 }
 printf("%d ", 199);
 total_head_movements += 199 - head_pos;
 head_pos = 199;
 for(i=0; i<num_blocks; i++){
 if(requests[i] <= head_pos){
 printf("%d ", requests[i]);
 total_head_movements += head_pos - requests[i];
 head_pos = requests[i];
 }
 }
 }
 printf("\nTotal number of head movements: %d\n", total_head_movements);
 return 0;
}
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)
-->
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define ARRAY_SIZE 1000
int main(int argc, char** argv) {
 int my_rank, num_processes;
 int* array;
 int max_num, local_max;
 int i;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
 MPI_Comm_size(MPI_COMM_WORLD, &num_processes);
 srand(time(NULL) + my_rank);
 array = (int*) malloc(ARRAY_SIZE / num_processes * sizeof(int));
 for (i = 0; i < ARRAY_SIZE / num_processes; i++) {
 array[i] = rand() % 1000;
 }
```

```
 local_max = array[0];
 for (i = 1; i < ARRAY_SIZE / num_processes; i++) {
 if (array[i] > local_max) {
 local_max = array[i];
 }
 }
 MPI_Reduce(&local_max, &max_num, 1, MPI_INT, MPI_MAX, 0,
MPI_COMM_WORLD);
 if (my_rank == 0) {
 printf("The maximum number is %d\n", max_num);
 }
 free(array);
 MPI_Finalize();
 return 0;
}
```

Slip 21

Q.1 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

```
-->
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int main(){
 int num_blocks, head_pos, total_head_movements=0, current_pos, i;
 printf("Enter the number of disk blocks: ");
 scanf("%d", &num_blocks);
 int requests[num_blocks];
 printf("Enter the disk request string: ");
 for(i=0; i<num_blocks; i++){
 scanf("%d", &requests[i]);
 }
 printf("Enter the current head position: ");
 scanf("%d", &head_pos);
 printf("\nOrder of served requests: ");
 // FCFS algorithm implementation
 for(i=0; i<num_blocks; i++){
 current_pos = requests[i];
 total_head_movements += abs(current_pos - head_pos);
 head_pos = current_pos;
 printf("%d ", current_pos);
 }
 printf("\nTotal number of head movements: %d\n", total_head_movements);
return 0;
}
```

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

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

```
#include <stdlib.h>
#include <time.h>
#define ARRAY_SIZE 1000
int main(int argc, char** argv) {
 int my_rank, num_processes;
 int* array;
 int local_sum = 0, even_sum = 0;
 int i;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
 MPI_Comm_size(MPI_COMM_WORLD, &num_processes);
 srand(time(NULL) + my_rank);
 array = (int*) malloc(ARRAY_SIZE / num_processes * sizeof(int));
 for (i = 0; i < ARRAY_SIZE / num_processes; i++) {
 array[i] = rand() % 1000;
 if (array[i] % 2 == 0) {
 local_sum += array[i];
 }
 }
 MPI_Reduce(&local_sum, &even_sum, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);
 if (my_rank == 0) {
 printf("The sum of all even numbers is %d\n", even_sum);
 }
 free(array);
 MPI_Finalize();
 return 0;
}
```

Slip 22

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

```
-->
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define ARRAY_SIZE 1000
int main(int argc, char** argv) {
 int my_rank, num_processes;
 int* array;
 int local_sum = 0, odd_sum = 0;
 int i;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
 MPI_Comm_size(MPI_COMM_WORLD, &num_processes);
 srand(time(NULL) + my_rank);
 array = (int*) malloc(ARRAY_SIZE / num_processes * sizeof(int));
 for (i = 0; i < ARRAY_SIZE / num_processes; i++) {
 array[i] = rand() % 1000;
 if (array[i] % 2 == 1) {
 local_sum += array[i];
 }
 }
 MPI_Reduce(&local_sum, &odd_sum, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);
 if (my_rank == 0) {
 printf("The sum of all odd numbers is %d\n", odd_sum);
```

```
 }
 free(array);
 MPI_Finalize();
 return 0;
}
```

Q.2 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
- Delete already created file
- Exit

```
-->
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX_BLOCKS 100
int disk[MAX_BLOCKS];
int num_blocks;
int free_blocks;
int *directory;
void init_disk() {
 for (int i = 0; i < num_blocks; i++) {
 disk[i] = rand() % 2; // randomly mark blocks as allocated or free
 if (disk[i] == 0) {
 free_blocks++;
 }
 }
}
void init_directory() {
 directory = (int *)malloc(num_blocks * sizeof(int));
 for (int i = 0; i < num_blocks; i++) {
 directory[i] = -1; // initialize directory with -1
 }
}
void show_bit_vector() {
 printf("Bit Vector:\n");
 for (int i = 0; i < num_blocks; i++) {
 printf("%d ", disk[i]);
 }
 printf("\n");
}
void show_directory() {
 printf("Directory:\n");
 for (int i = 0; i < num_blocks; i++) {
 printf("%d ", directory[i]);
 }
 printf("\n");
}
void delete_file(int start_block) {
 int current_block = start_block;
 while (current_block != -1) {
 disk[current_block] = 0;
 free_blocks++;
 current_block = directory[current_block];
 directory[current_block] = -1;
 }
 printf("File deleted successfully.\n");
}
int main() {
 printf("Enter the number of blocks in the disk: ");
 scanf("%d", &num_blocks);
 init_disk();
```

```
 init_directory();
 int choice, start_block;
 bool running = true;
 while (running) {
 printf("\n1. Show Bit Vector\n2. Show Directory\n3. Delete File\n4. Exit\n");
 printf("Enter your choice: ");
 scanf("%d", &choice);
 switch (choice) {
 case 1:
 show_bit_vector();
 break;
 case 2:
 show_directory();
 break;
 case 3:
 printf("Enter the starting block of the file: ");
 scanf("%d", &start_block);
 delete_file(start_block);
 break;
 case 4:
 running = false;
 break;
 default:
 printf("Invalid choice. Try again.\n");
 }
 }
 return 0;
}
```

Slip 23

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
-->

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_RESOURCES 10
#define MAX_PROCESSES 10
int available[MAX_RESOURCES];
int allocation[MAX_PROCESSES][MAX_RESOURCES];
int maximum[MAX_PROCESSES][MAX_RESOURCES];
int need[MAX_PROCESSES][MAX_RESOURCES];
int main() {
 int m, n, i, j;
 printf("Enter the number of processes: ");
 scanf("%d", &m);
 printf("Enter the number of resources: ");
 scanf("%d", &n);
 printf("Enter the number of available resources of each type:\n");
 for (i = 0; i < n; i++) {
 printf("Resource %d: ", i);
 scanf("%d", &available[i]);
```

```
 }
 printf("Enter the allocation matrix:\n");
 for (i = 0; i < m; i++) {
 printf("Process %d: ", i);
 for (j = 0; j < n; j++) {
 scanf("%d", &allocation[i][j]);
 }
 }
 printf("Enter the maximum requirement matrix:\n");
 for (i = 0; i < m; i++) {
 printf("Process %d: ", i);
 for (j = 0; j < n; j++) {
 scanf("%d", &maximum[i][j]);
 }
 }
 // Calculate the need matrix
 for (i = 0; i < m; i++) {
 for (j = 0; j < n; j++) {
 need[i][j] = maximum[i][j] - allocation[i][j];
 }
 }
 // Display the need matrix
 printf("Need matrix:\n");
 for (i = 0; i < m; i++) {
 printf("Process %d: ", i);
 for (j = 0; j < n; j++) {
 printf("%d ", need[i][j]);
 }
 printf("\n");
 }
 // Check if the request can be granted immediately
 int process;
 printf("Enter the process number making the request: ");
 scanf("%d", &process);
 int request[MAX_RESOURCES];
 printf("Enter the resource request vector:\n");
 for (i = 0; i < n; i++) {
 scanf("%d", &request[i]);
 }
 for (i = 0; i < n; i++) {
 if (request[i] > need[process][i]) {
 printf("Error: Process %d exceeded its maximum claim\n", process);
 return 0;
 }
 if (request[i] > available[i]) {
 printf("Process %d must wait since resources are not available\n", process);
 return 0;
 }
 }
 printf("Request can be granted immediately\n");
 return 0;
}
```

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.
24, 90, 133, 43, 188, 70, 37, 55
Start Head Position: 58
-->

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

```
int main() {
 int n, head, i, j, min, min_index, total_head_movements = 0;

 printf("Enter the total number of disk blocks: ");
 scanf("%d", &n);

 int requests[n], completed[n];

 printf("Enter the disk request string: ");
 for(i=0; i<n; i++) {
 scanf("%d", &requests[i]);
 completed[i] = 0; // Initialize all requests as incomplete
 }

 printf("Enter the current head position: ");
 scanf("%d", &head);

 printf("\nOrder of disk request served: ");

 for(i=0; i<n; i++) {
 min = __INT_MAX__;
 for(j=0; j<n; j++) {
 if(completed[j] == 1) {
 continue; // Skip completed requests
 }

 if(abs(head - requests[j]) < min) {
 min = abs(head - requests[j]);
 min_index = j;
 }
 }

 completed[min_index] = 1; // Mark request as completed
 total_head_movements += min; // Add distance to total head movements
 head = requests[min_index]; // Move head to the selected request
 printf("%d ", requests[min_index]);
 }

 printf("\nTotal head movements: %d", total_head_movements);

 return 0;
}
```

Slip 24

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

```
-->
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define ARRAY_SIZE 1000
int main(int argc, char** argv) {
 int my_rank, num_processes;
 int* array;
 int local_sum = 0, odd_sum = 0;
 int i;
```

```
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
 MPI_Comm_size(MPI_COMM_WORLD, &num_processes);
 srand(time(NULL) + my_rank);
 array = (int*) malloc(ARRAY_SIZE / num_processes * sizeof(int));
 for (i = 0; i < ARRAY_SIZE / num_processes; i++) {
 array[i] = rand() % 1000;
 if (array[i] % 2 == 1) {
 local_sum += array[i];
 }
 }
 MPI_Reduce(&local_sum, &odd_sum, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);
 if (my_rank == 0) {
 printf("The sum of all odd numbers is %d\n", odd_sum);
 }
 free(array);
 MPI_Finalize();
 return 0;
}
```

Q.2 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.

-->

```
#include <stdio.h>
#define N 5
#define M 4
int main()
{
 int allocation[N][M] = {{0, 0, 1, 2},
 {1, 0, 0, 0},
 {1, 3, 5, 4},
 {0, 6, 3, 2},
 {0, 0, 1, 4}};
 int max[N][M] = {{0, 0, 1, 2},
 {1, 7, 5, 0},
 {2, 3, 5, 6},
 {0, 6, 5, 2},
 {0, 6, 5, 6}};
 int available[M] = {1, 5, 2, 0};
 int need[N][M];
 // Calculate need matrix
 for (int i = 0; i < N; i++) {
 for (int j = 0; j < M; j++) {
 need[i][j] = max[i][j] - allocation[i][j];
 }
 }
 // Initialize finish array
 int finish[N] = {0};
 // Initialize safe sequence array
 int safe_seq[N];
 // Initialize work array
 int work[M];
 for (int i = 0; i < M; i++) {
 work[i] = available[i];
```

```
 }
 // Find safe sequence
 int count = 0;
 while (count < N) {
 int found = 0;
 for (int i = 0; i < N; i++) {
 if (finish[i] == 0) {
 int j;
 for (j = 0; j < M; j++) {
 if (need[i][j] > work[j])
 break;
 }
 if (j == M) {
 for (int k = 0; k < M; k++) {
 work[k] += allocation[i][k];
 }
 safe_seq[count++] = i;
 finish[i] = 1;
 found = 1;
 }
 }
 }
 if (found == 0) {
 printf("System is not in safe state.\n");
 return 0;
 }
 }
 // Print need matrix
 printf("Need matrix:\n");
 for (int i = 0; i < N; i++) {
 printf("P%d: ", i);
 for (int j = 0; j < M; j++) {
 printf("%d ", need[i][j]);
 }
 printf("\n");
 }
 // Print safe sequence
 printf("System is in safe state.\nSafe sequence is: ");
 for (int i = 0; i < N; i++) {
 printf("P%d ", safe_seq[i]);
 }
 printf("\n");
 return 0;
}
```

Slip 25

Q.1 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.
86, 147, 91, 170, 95, 130, 102, 70
Starting Head position= 125
Direction: User Defined
-->

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

```
int main() {
 int n, head, i, j, total_head_movements = 0;
 char direction;

 printf("Enter the total number of disk blocks: ");
 scanf("%d", &n);
 int requests[n];
 printf("Enter the disk request string: ");
 for(i=0; i<n; i++) {
 scanf("%d", &requests[i]);
 }

 printf("Enter the current head position: ");
 scanf("%d", &head);

 printf("Enter the direction (L/R): ");
 scanf(" %c", &direction);

 // Sort the requests in ascending order
 for(i=0; i<n-1; i++) {
 for(j=0; j<n-i-1; j++) {
 if(requests[j] > requests[j+1]) {
 int temp = requests[j];
 requests[j] = requests[j+1];
 requests[j+1] = temp;
 }
 }
 }

 printf("\nOrder of disk request served: ");

 if(direction == 'L') {
 // Scan left from the head
 for(i=0; i<n; i++) {
 if(requests[i] >= head) {
 break; // Stop at first request greater than or equal to head
 }
 }

 for(j=i; j>=0; j--) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }

 for(j=i+1; j<n; j++) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }
 }
 else {
 // Scan right from the head
 for(i=n-1; i>=0; i--) {
 if(requests[i] <= head) {
 break; // Stop at first request less than or equal to head
 }
 }

 for(j=i; j<n; j++) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }
```

```
 for(j=i-1; j>=0; j--) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }
 }

 printf("\nTotal head movements: %d", total_head_movements);

 return 0;
}
```

Q.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.

- Show Bit Vector
- Create New File
- Show Directory
- 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<fb;i++)
bit[i] = 0;
printf("\n");
}
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\t\t\tNumber of blocks\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,k,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;
}
```

Slip 26

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.
-->

```
#include <stdio.h>
#define N 5
#define M 4
int main()
{
 int allocation[N][M] = {{0, 0, 1, 2},
 {1, 0, 0, 0},
 {1, 3, 5, 4},
 {0, 6, 3, 2},
 {0, 0, 1, 4}};
 int max[N][M] = {{0, 0, 1, 2},
 {1, 7, 5, 0},
 {2, 3, 5, 6},
 {0, 6, 5, 2},
 {0, 6, 5, 6}};
 int available[M] = {1, 5, 2, 0};
 int need[N][M];
 // Calculate need matrix
 for (int i = 0; i < N; i++) {
 for (int j = 0; j < M; j++) {
 need[i][j] = max[i][j] - allocation[i][j];
 }
 }
 // Initialize finish array
 int finish[N] = {0};
 // Initialize safe sequence array
 int safe_seq[N];
 // Initialize work array
 int work[M];
 for (int i = 0; i < M; i++) {
 work[i] = available[i];
 }
 // Find safe sequence
 int count = 0;
 while (count < N) {
 int found = 0;
 for (int i = 0; i < N; i++) {
 if (finish[i] == 0) {
 int j;
 for (j = 0; j < M; j++) {
 if (need[i][j] > work[j])
 break;
 }
 if (j == M) {
 for (int k = 0; k < M; k++) {
 work[k] += allocation[i][k];
 }
 safe_seq[count++] = i;
 finish[i] = 1;
 found = 1;
 }
 }
 }
```

```
 if (found == 0) {
 printf("System is not in safe state.\n");
 return 0;
 }
 }
 // Print need matrix
 printf("Need matrix:\n");
 for (int i = 0; i < N; i++) {
 printf("P%d: ", i);
 for (int j = 0; j < M; j++) {
 printf("%d ", need[i][j]);
 }
 printf("\n");
 }
 // Print safe sequence
 printf("System is in safe state.\nSafe sequence is: ");
 for (int i = 0; i < N; i++) {
 printf("P%d ", safe_seq[i]);
 }
 printf("\n");
 return 0;
}
```

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.
 56, 59, 40, 19, 91, 161, 151, 39, 185
 Start Head Position: 48

```
-->
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int main(){
 int num_blocks, head_pos, total_head_movements=0, current_pos, i;
 printf("Enter the number of disk blocks: ");
 scanf("%d", &num_blocks);
 int requests[num_blocks];
 printf("Enter the disk request string: ");
 for(i=0; i<num_blocks; i++){
 scanf("%d", &requests[i]);
 }
 printf("Enter the current head position: ");
 scanf("%d", &head_pos);
 printf("\nOrder of served requests: ");
 // FCFS algorithm implementation
 for(i=0; i<num_blocks; i++){
 current_pos = requests[i];
 total_head_movements += abs(current_pos - head_pos);
 head_pos = current_pos;
 printf("%d ", current_pos);
 }
```

Slip 27

Q.1 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.

176, 79, 34, 60, 92, 11, 41, 114

Starting Head Position: 65

Direction: Right

-->

```
#include <stdio.h>
#include <stdlib.h>
int main() {
 int n, head, i, j, total_head_movements = 0;
 char direction;

 printf("Enter the total number of disk blocks: ");
 scanf("%d", &n);

 int requests[n];

 printf("Enter the disk request string: ");
 for(i=0; i<n; i++) {
 scanf("%d", &requests[i]);
 }

 printf("Enter the current head position: ");
 scanf("%d", &head);

 printf("Enter the direction (L/R): ");
 scanf(" %c", &direction);

 // Sort the requests in ascending order
 for(i=0; i<n-1; i++) {
 for(j=0; j<n-i-1; j++) {
 if(requests[j] > requests[j+1]) {
 int temp = requests[j];
 requests[j] = requests[j+1];
 requests[j+1] = temp;
 }
 }
 }

 printf("\nOrder of disk request served: ");

 if(direction == 'L') {
 // Scan left from the head
 for(i=0; i<n; i++) {
 if(requests[i] >= head) {
 break; // Stop at first request greater than or equal to head
 }
 }

 for(j=i; j>=0; j--) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }

 for(j=i+1; j<n; j++) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }
```

```
 }
 else {
 // Scan right from the head
 for(i=n-1; i>=0; i--) {
 if(requests[i] <= head) {
 break; // Stop at first request less than or equal to head
 }
 }

 for(j=i; j<n; j++) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }

 for(j=i-1; j>=0; j--) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }
 }

 printf("\nTotal head movements: %d", total_head_movements);
 return 0;
}
```

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)

-->

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>
#define ARRAY_SIZE 1000
int main(int argc, char** argv) {
 int my_rank, num_processes;
 int* array;
 int local_min = INT_MAX, global_min;
 int i;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
 MPI_Comm_size(MPI_COMM_WORLD, &num_processes);
 srand(time(NULL) + my_rank);
 array = (int*) malloc(ARRAY_SIZE / num_processes * sizeof(int));
 for (i = 0; i < ARRAY_SIZE / num_processes; i++) {
 array[i] = rand() % 1000;
 if (array[i] < local_min) {
 local_min = array[i];

 }
 }
 MPI_Reduce(&local_min, &global_min, 1, MPI_INT, MPI_MIN, 0,
MPI_COMM_WORLD);
 if (my_rank == 0) {
 printf("The minimum number is %d\n", global_min);
 }
 free(array);
 MPI_Finalize();
 return 0;
}
```

Slip 28

Q.1 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.
56, 59, 40, 19, 91, 161, 151, 39, 185
Start Head Position: 48
Direction: User Defined
-->

```
#include <stdio.h>
#include <stdlib.h>
int main() {
 int n, head, i, j, total_head_movements = 0;
 char direction;

 printf("Enter the total number of disk blocks: ");
 scanf("%d", &n);
 int requests[n];

 printf("Enter the disk request string: ");
 for(i=0; i<n; i++) {
 scanf("%d", &requests[i]);
 }

 printf("Enter the current head position: ");
 scanf("%d", &head);

 printf("Enter the direction (L/R): ");
 scanf(" %c", &direction);

 // Sort the requests in ascending order
 for(i=0; i<n-1; i++) {
 for(j=0; j<n-i-1; j++) {
 if(requests[j] > requests[j+1]) {
 int temp = requests[j];
 requests[j] = requests[j+1];
 requests[j+1] = temp;
 }
 }
 }

 printf("\nOrder of disk request served: ");

 if(direction == 'L') {
 // Scan left from the head
 for(i=n-1; i>=0; i--) {
 if(requests[i] <= head) {
 break; // Stop at first request less than or equal to head
 }
 }


 for(j=i; j>=0; j--) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
```

```
 head = requests[j];
 }

 for(j=n-1; j>i; j--) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }
 }
 else {
 // Scan right from the head
 for(i=0; i<n; i++) {
 if(requests[i] >= head) {
 break; // Stop at first request greater than or equal to head
 }
 }

 for(j=i; j<n; j++) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }

 for(j=0; j<i; j++) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }
 }


 printf("\nTotal head movements: %d", total_head_movements);

 return 0;
}
Q.2 Write an MPI program to calculate sum of randomly generated 1000 numbers
(stored in array) on a cluster
-->
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define ARRAY_SIZE 1000
int main(int argc, char** argv) {
 int my_rank, num_processes;
 int* array;
 int max_num, local_max;
 int i;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
 MPI_Comm_size(MPI_COMM_WORLD, &num_processes);
 srand(time(NULL) + my_rank);
 array = (int*) malloc(ARRAY_SIZE / num_processes * sizeof(int));
 for (i = 0; i < ARRAY_SIZE / num_processes; i++) {
 array[i] = rand() % 1000;
 }
 local_max = array[0];
 for (i = 1; i < ARRAY_SIZE / num_processes; i++) {
 if (array[i] > local_max) {
 local_max = array[i];
 }
 }
 MPI_Reduce(&local_max, &max_num, 1, MPI_INT, MPI_MAX, 0,
MPI_COMM_WORLD);
```

```
 if (my_rank == 0) {
 printf("The maximum number is %d\n", max_num);
 }
 free(array);
 MPI_Finalize();
 return 0;
}
```

Slip 29

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

```
-->
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define ARRAY_SIZE 1000
int main(int argc, char** argv) {
 int my_rank, num_processes;
 int* array;

 int local_sum = 0, even_sum = 0;
 int i;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
 MPI_Comm_size(MPI_COMM_WORLD, &num_processes);
 srand(time(NULL) + my_rank);
 array = (int*) malloc(ARRAY_SIZE / num_processes * sizeof(int));
 for (i = 0; i < ARRAY_SIZE / num_processes; i++) {
 array[i] = rand() % 1000;
 if (array[i] % 2 == 0) {
 local_sum += array[i];
 }
 }
 MPI_Reduce(&local_sum, &even_sum, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);
 if (my_rank == 0) {
 printf("The sum of all even numbers is %d\n", even_sum);
 }
 free(array);
 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.. [15]
80, 150, 60,135, 40, 35, 170
Starting Head Position: 70
Direction: Right

```
-->
#include <stdio.h>
#include <stdlib.h>
int main() {
 int n, head, i, j, total_head_movements = 0;
 char direction;

 printf("Enter the total number of disk blocks: ");
 scanf("%d", &n);

 int requests[n];

 printf("Enter the disk request string: ");
 for(i=0; i<n; i++) {
 scanf("%d", &requests[i]);
 }

 printf("Enter the current head position: ");
 scanf("%d", &head);

 printf("Enter the direction (L/R): ");
 scanf(" %c", &direction);

 // Sort the requests in ascending order
 for(i=0; i<n-1; i++) {
 for(j=0; j<n-i-1; j++) {
 if(requests[j] > requests[j+1]) {
 int temp = requests[j];
 requests[j] = requests[j+1];
 requests[j+1] = temp;
 }
 }
 }

 printf("\nOrder of disk request served: ");

 if(direction == 'L') {
 // Scan left from the head
 for(i=n-1; i>=0; i--) {
 if(requests[i] <= head) {
 break; // Stop at first request less than or equal to head
 }
 }

 for(j=i; j>=0; j--) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }

 for(j=n-1; j>i; j--) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }
 }
 else {
 // Scan right from the head
 for(i=0; i<n; i++) {
 if(requests[i] >= head) {
 break; // Stop at first request greater than or equal to head
 }
 }
```

```
 for(j=i; j<n; j++) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }

 for(j=0; j<i; j++) {
 printf("%d ", requests[j]);
 total_head_movements += abs(head - requests[j]);
 head = requests[j];
 }
 }

 printf("\nTotal head movements: %d", total_head_movements);

 return 0;
}
```

Slip 30

Q.1 Write an MPI program to find the min number from randomly generated 1000 numbers
(stored in array) on a cluster (Hint: Use MPI_Reduce)
-->

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>
#define ARRAY_SIZE 1000
int main(int argc, char** argv) {
 int my_rank, num_processes;
 int* array;
 int local_min = INT_MAX, global_min;
 int i;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
 MPI_Comm_size(MPI_COMM_WORLD, &num_processes);
 srand(time(NULL) + my_rank);
 array = (int*) malloc(ARRAY_SIZE / num_processes * sizeof(int));
 for (i = 0; i < ARRAY_SIZE / num_processes; i++) {
 array[i] = rand() % 1000;
 if (array[i] < local_min) {
 local_min = array[i];
 }
 }
 MPI_Reduce(&local_min, &global_min, 1, MPI_INT, MPI_MIN, 0,
MPI_COMM_WORLD);
 if (my_rank == 0) {
 printf("The minimum number is %d\n", global_min);
 }
 free(array);
 MPI_Finalize();
 return 0;
}
```

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.
 65, 95, 30, 91, 18, 116, 142, 44, 168
 Start Head Position: 52
-->
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int main(){
 int num_blocks, head_pos, total_head_movements=0, current_pos, i;
 printf("Enter the number of disk blocks: ");
 scanf("%d", &num_blocks);
 int requests[num_blocks];
 printf("Enter the disk request string: ");
 for(i=0; i<num_blocks; i++){
 scanf("%d", &requests[i]);
 }
 printf("Enter the current head position: ");
 scanf("%d", &head_pos);
 printf("\nOrder of served requests: ");
 // FCFS algorithm implementation
 for(i=0; i<num_blocks; i++){
 current_pos = requests[i];
 total_head_movements += abs(current_pos - head_pos);
 head_pos = current_pos;
 printf("%d ", current_pos);
 }
 printf("\nTotal number of head movements: %d\n", total_head_movements);
 return 0;
}
```