Slip 1

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

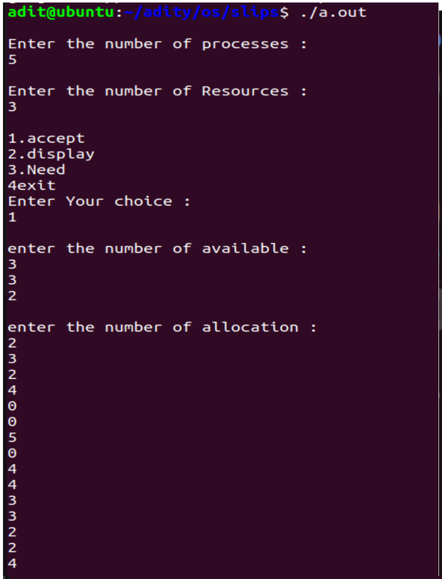
d) Display Available

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			

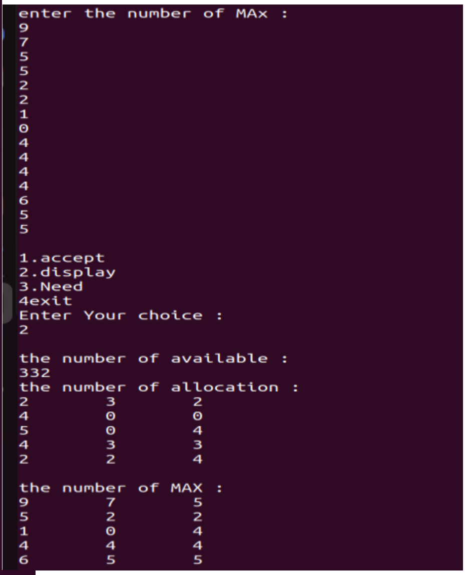
```
#include<stdio.h>
#include<stdlib.h>
int available[20],Need[20][20],MAX[20][20],alloction[20][20],n,m;
void accept_matrix(int arr[20][20])
{
        int i,j;
        for(i=0;i<n;i++)
        {
                for(j=0;j<m;j++)
                {
                        scanf("%d",&arr[i][j]);
                }
        }
}
void accept_array(int arr[20],int no)
{
        int i;
        for(i=0;i<no;i++)
        {

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

        }
}
void display_matrix(int arr[20][20])
{
        int i,j;
        for(i=0;i<n;i++)
        {
                for(j=0;j<m;j++)
                {
                        printf("%d\t", arr[i][j]);
                }
                printf("\n");
        }
}
```

```
void display_array(int arr[20],int no)
{
    int i;
    for(i=0;i<no;i++)
    {

            printf("%d", arr[i]);

    }
}
void find_need()
{
    int i,j;
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        {
            Need[i][j]=MAX[i][j] - alloction[i][j];
        }
    }
}

void main()
{
    printf("\nEnter the number of processes :\n");
    scanf("%d",&n);
    printf("\nEnter the number of Resources :\n");
    scanf("%d",&m);
    int ch;
    while(ch!=6)
    {


        printf("\n1.accept\n2.display\n3.Need\n4exit\n");
        printf("Enter Your choice : \n");
        scanf("%d", &ch);

        switch(ch)
        {

        case 1:
            printf("\nenter the number of available :\n");
            accept_array(available,m);
            printf("\nenter the number of allocation :\n");
            accept_matrix(alloction);
            printf("\nenter the number of MAx :\n");
            accept_matrix(MAX);
            break;
        case 2:

            printf("\nthe number of available :\n");
```

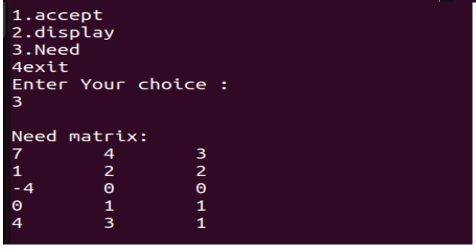
```
                        display_array(available,m);
                        printf("\nthe number of allocation :\n");
                        display_matrix(alloction);
                        printf("\nthe number of MAX :\n");
                        display_matrix(MAX);
                        break;
                case 3:
                        find_need();
                        printf("\nNeed matrix:\n");
                        display_matrix(Need);
                        break;
                case 4:
                        printf("exit\n");
                        break;
                default:
                        printf("invalid choice\n");
                }
        }
}
```



```
adit@ubuntu:~/adity/os/slips$ ./a.out

Enter the number of processes :
5

Enter the number of Resources :
3

1.accept
2.display
3.Need
4exit
Enter Your choice :
1

enter the number of available :
3
3
2

enter the number of allocation :
2
3
2
4
0
0
5
0
4
4
3
3
2
2
4
```



```
enter the number of MAx :
9
7
5
5
2
2
1
0
4
4
4
4
6
5
5

1.accept
2.display
3.Need
4exit
Enter Your choice :
2

the number of available :
332
the number of allocation :
2       3       2
4       0       0
5       0       4
4       3       3
2       2       4

the number of MAX :
9       7       5
5       2       2
1       0       4
4       4       4
6       5       5
```

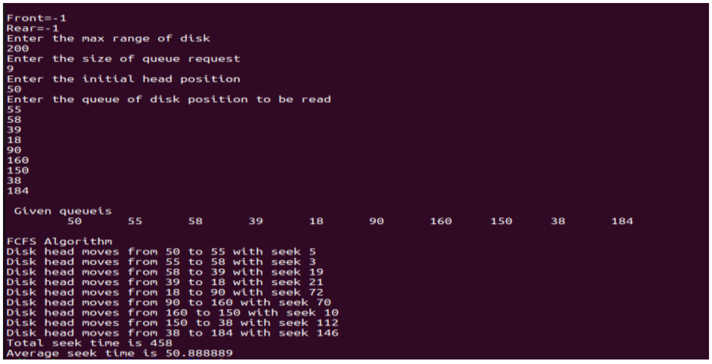


```
1.accept
2.display
3.Need
4exit
Enter Your choice :
3

Need matrix:
7       4       3
1       2       2
-4      0       0
0       1       1
4       3       1
```

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

```
#include<stdio.h>
#include<stdlib.h>
int front,rear;
void init()
```

```
{
        front = rear = -1;
}
void display(int *Q)
{
        int i;
        for(i=front;i<=rear;i++)
        printf("\t%d",Q[i]);
}
void enqueue(int *Q,int n,int var)
{
        if(rear==n)
        printf("\nQueue is full");
        else
        {
                if(front==-1)
                        front++;
                        rear++;
                        Q[rear]=var;
        }
}
int FCFS(int *Q,int n)
{
        int j,seek=0,diff;
        for(j=0;j<n;j++)
        {
                diff = abs(Q[j+1]-Q[j]);
                seek+=diff;
                printf("Disk head moves from %d to %d with seek %d\n",Q[j],Q[j+1],diff);
        }
return seek;
}
int main()
{
int queue[20],n,var,head,i,j,k,seek,max;
        float avg;
        init();
printf("\nFront=%d\nRear=%d",front,rear);

        printf("\nEnter the max range of disk\n");
        scanf("%d",&max);
        printf("Enter the size of queue request\n");
        scanf("%d",&n);
        printf("Enter the initial head position\n");
        scanf("%d",&head);
        printf("Enter the queue of disk position to be read\n");
        enqueue(queue,n,head);
        for(i=1;i<=n;i++)
        {
                scanf("%d",&var);
                if(var<0||var>max)
                printf("\n Error ..!given position is invalid\n");
                else
                {
                enqueue(queue,n,var);
                }
        }
        printf("\n Given queueis\n");
        display(queue);
        printf("\n\nFCFS Algorithm\n");
        seek = FCFS(queue,n);
        printf("Total seek time is %d\n",seek);
        avg=seek/(float)n;
printf("Average seek time is %f\n",avg);
        return 0;
}
```



```
Front=-1
Rear=-1
Enter the max range of disk
200
Enter the size of queue request
9
Enter the initial head position
50
Enter the queue of disk position to be read
55
58
39
18
90
160
150
38
184

 Given queueis
       50      55      58      39      18      90      160     150     38      184

FCFS Algorithm
Disk head moves from 50 to 55 with seek 5
Disk head moves from 55 to 58 with seek 3
Disk head moves from 58 to 39 with seek 19
Disk head moves from 39 to 18 with seek 21
Disk head moves from 18 to 90 with seek 72
Disk head moves from 90 to 160 with seek 70
Disk head moves from 160 to 150 with seek 10
Disk head moves from 150 to 38 with seek 112
Disk head moves from 38 to 184 with seek 146
Total seek time is 458
Average seek time is 50.888889
```

Slip 2

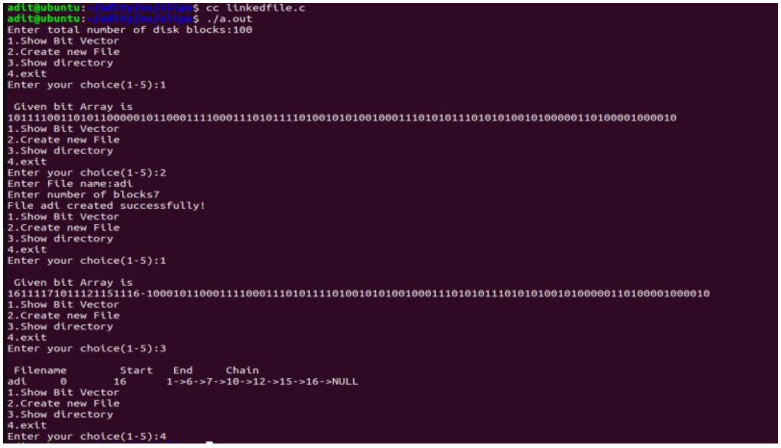
Q.1Write a program to simulate Linked file allocation method. Assume disk with n number of blocks. Give value of n as input. Randomly mark some block as allocated and accordingly maintain the list of free blocks Write menu driver program with menu options as mentioned below and implement each option.

- 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,s[10],l[10],k=0,fb,bit[MAX];
void Generate_BitVector()
{
        int i,num;
```

```
		for(i=0;i<n;i++)
		{
			num=rand();
			bit[i]=num%2;
		}
}

void show_bitVector()
{
		int i;
		printf("\n Given bit Array is\n");
		for(i=0;i<=n;i++)
		{
			printf("%d",bit[i]);
		}
		printf("\n");
}

void showDirectory()
{
		NODE *p;
		int j,i;
		printf("\n Filename\t Start\t End\t Chain\n");
		p=first;
		j=0;
		while(p!=NULL)
		{
			printf("%s\t%d\t%d\t",p->fname,s[j],l[j]);
			i=p->start;
			while(i!=-1)
			{
				printf("%d->",i);
				i=bit[i];
			}
			printf("NULL\n");
			p=p->next;
			j++;
		}
}
void create()
{
		NODE *p;
		char fname[20];
		int i,j,nob;
		printf("Enter File name:");
		scanf("%s",fname);
		printf("Enter number of blocks");
		scanf("%d",&nob);
		if(nob>fb)
		{
			printf("Failed to create %s file",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;
        l[k]=i;
        k++;
        printf("File %s created successfully!\n",fname);
}
int main()
{
        int ch;
        printf("Enter total number of disk blocks:");
        scanf("%d",&n);
        fb=n;
        Generate_BitVector();
        while(1)
        {
                printf("1.Show Bit Vector\n");
                printf("2.Create new File\n");
                printf("3.Show directory\n");
                printf("4.exit\n");
                printf("Enter your choice(1-5):");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                                show_bitVector();
                                break;
                        case 2:
                                create();
                                break;
                        case 3:
                                showDirectory();
                                break;
                        case 4:
                                exit(0);
                }
        }
        return 0;
}
```

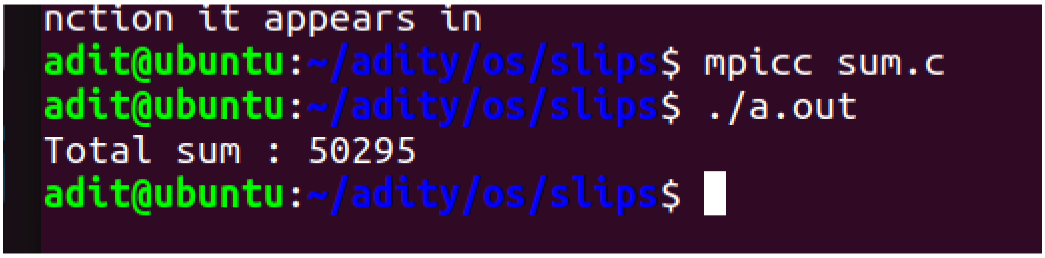


```
adit@ubuntu:~/adity/os/slips$ cc linkedfile.c
adit@ubuntu:~/adity/os/slips$ ./a.out
Enter total number of disk blocks:100
1.Show Bit Vector
2.Create new File
3.Show directory
4.exit
Enter your choice(1-5):1

 Given bit Array is
1011110011010110000001011000111100011101011110100101010010001110101011101010100101000001101000010000 10
1.Show Bit Vector
2.Create new File
3.Show directory
4.exit
Enter your choice(1-5):2
Enter File name:adi
Enter number of blocks7
File adi created successfully!
1.Show Bit Vector
2.Create new File
3.Show directory
4.exit
Enter your choice(1-5):1

 Given bit Array is
16111171011121151116-1000101100011110001110101111010010101001000111010101110101010010100000110100001000010
1.Show Bit Vector
2.Create new File
3.Show directory
4.exit
Enter your choice(1-5):3

 Filename        Start   End     Chain
adi     0       16      1->6->7->10->12->15->16->NULL
1.Show Bit Vector
2.Create new File
3.Show directory
4.exit
Enter your choice(1-5):4
```

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

```
#include<stdio.h>
#include<stdlib.h>
#include<mpi.h>
#define ARRAY_SIZE 1000
int main(int argc,char *argv[])
{
        int rank,size,partial_sum=0,total_sum=0,number[ARRAY_SIZE];

        MPI_Init(&argc,&argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        MPI_Comm_size(MPI_COMM_WORLD, &size);

        srand(rank);
        for(int i=0;i<ARRAY_SIZE;i++)
        {
                number[i]=rand()%100;
        }

        for(int i=0;i<ARRAY_SIZE;i++)
        {
```

```
        partial_sum += number[i];
    }

    MPI_Reduce(&partial_sum , &total_sum,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
    if(rank == 0)
    {
        printf("Total sum : %d\n",total_sum);
    }
    MPI_Finalize();
    return 0;
}
```



```
nction it appears in
adit@ubuntu:~/adity/os/slips$ mpicc sum.c
adit@ubuntu:~/adity/os/slips$ ./a.out
Total sum : 50295
adit@ubuntu:~/adity/os/slips$
```

Slip 3

Q.1Write 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.

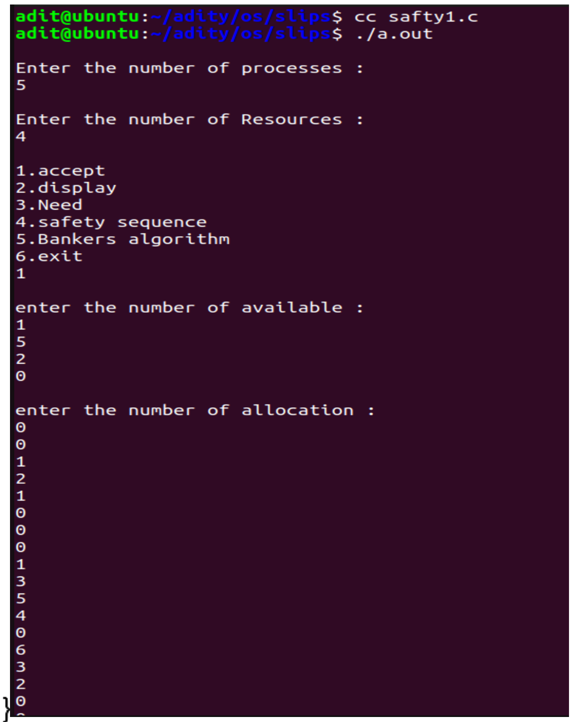
Process	Allocation		Max	Available								
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

a) Calculate and display the content of need matrix?

b) Is the system in safe state? If display the safe sequence.

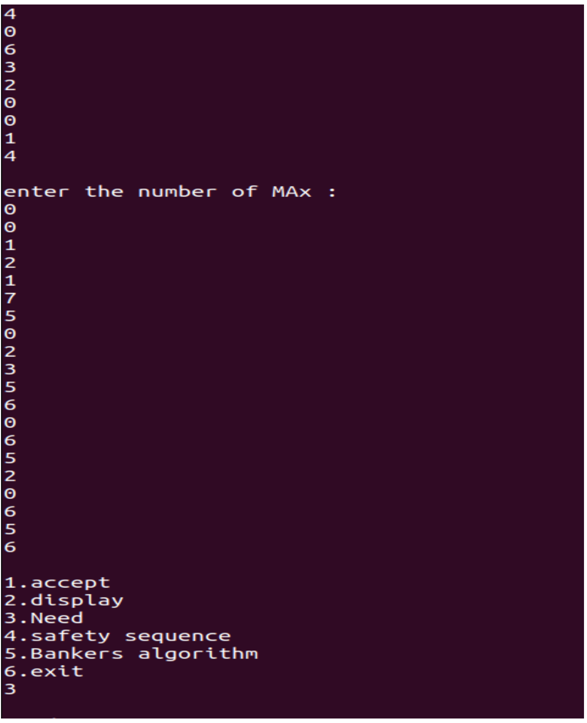
```
#include<stdio.h>
#include<stdlib.h>
int available[20],Need[20][20],MAX[20][20],alloction[20][20],work[20],safe[20],finish[20],n,m;
void accept_matrix(int arr[20][20])
{
    int i,j;
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        {
            scanf("%d",&arr[i][j]);
        }
    }
}
void accept_array(int arr[20],int no)
{
    int i;
    for(i=0;i<no;i++)
    {
            scanf("%d",&arr[i]);

    }
}
void display_matrix(int arr[20][20])
{
    int i,j;
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        {
            printf("%d\t", arr[i][j]);
        }
        printf("\n");
    }
}
void display_array(int arr[20],int no)
{
    int i;
    for(i=0;i<no;i++)
    {

            printf("%d", arr[i]);

    }
}
void find_need()
{
    int i,j;
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        {
            Need[i][j]=MAX[i][j] - alloction[i][j];
        }
    }
}
int compare_need(int p)
{
    int i,j,flag=0;
    for(j=0;j<m;j++)
    {
        if(Need[p][j]>work[j])
        {
            flag=1;
            break;
        }
    }
    if(flag==0)
    {
        return p;
    }
    return -1;
}
void safety_algo()
{
    int over=0,i,j,k,pno,l=0,flag;
    for(i=0;i<m;i++)
    work[i]=available[i];
    for(i=0;i<n;i++)
    finish[i]=0;
    while(!over)
    {
            for(i=0;i<n;i++)
            {
                if(finish[i]==0)
                {
                    flag=0;
                    pno=compare_need(i);
                    if(pno>-1)
                    break;
                }
            }
            if(i==n)
            {
                printf("system is not safe\n");
                exit(1);
            }
            if(i<n && pno>=0)
            {
                for(k=0;k<m;k++)
                work[k]+=alloction[pno][k];
                finish[pno]=1;
                safe[l++]=pno;
                if(l>=n)
                {
                    printf("\nsafe sequence is :\n");
                    for(l=0;l<n;l++)
                    {
                        printf("P%d\t",safe[l]);
                        over=1;
                    }
                }
            }
        }
}
void main()
{
    printf("\nEnter the number of processes :\n");
    scanf("%d",&n);
    printf("\nEnter the number of Resources :\n");
```

```
scanf("%d",&m);
int ch;
while(ch!=6)
{
        printf("\n1.accept\n2.display\n3.Need\n4.safety sequence\n5.exit\n");
        scanf("%d", &ch);
        switch(ch)
        {

        case 1:
                printf("\nenter the number of available :\n");
                accept_array(available,m);
                printf("\nenter the number of allocation :\n");
                accept_matrix(alloction);
                printf("\nenter the number of MAx :\n");
                accept_matrix(MAX);
                break;
        case 2:

                printf("\nthe number of available :\n");
                display_array(available,m);
                printf("\nthe number of allocation :\n");
                display_matrix(alloction);
                printf("\nthe number of MAX :\n");
                display_matrix(MAX);
                break;
        case 3:
                find_need();
                printf("\nNeed matrix:\n");
                display_matrix(Need);
                break;
        case 4:
                safety_algo();
                break;

        case 5:
                printf("exit\n");
                break;
        default:
                printf("invalid choice\n");
        }
}
```

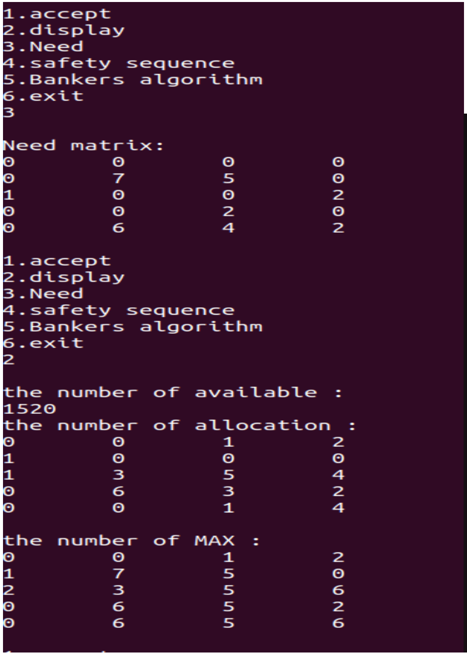


```
adit@ubuntu:~/adity/os/slips$ cc safty1.c
adit@ubuntu:~/adity/os/slips$ ./a.out

Enter the number of processes :
5

Enter the number of Resources :
4

1.accept
2.display
3.Need
4.safety sequence
5.Bankers algorithm
6.exit
1

enter the number of available :
1
5
2
0

enter the number of allocation :
0
0
1
2
1
0
0
0
1
3
5
4
0
6
3
2
0
```

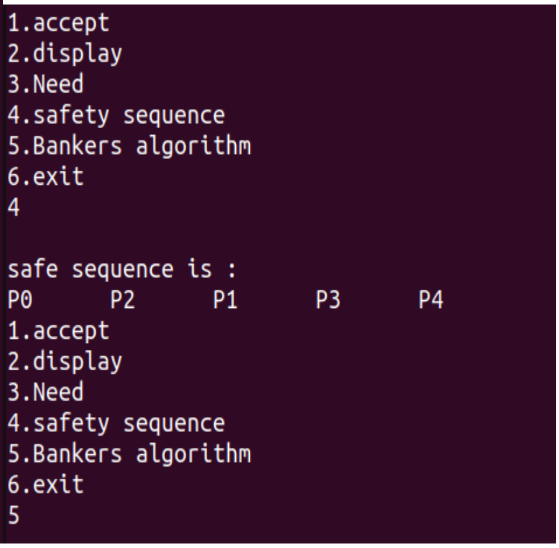
}



```
4
0
6
3
2
0
0
1
4

enter the number of MAx :
0
0
1
2
1
7
5
0
2
3
5
6
0
6
5
2
0
6
5
6

1.accept
2.display
3.Need
4.safety sequence
5.Bankers algorithm
6.exit
3
```



```
1.accept
2.display
3.Need
4.safety sequence
5.Bankers algorithm
6.exit
3

Need matrix:
0       0       0       0
0       7       5       0
1       0       0       2
0       0       2       0
0       6       4       2

1.accept
2.display
3.Need
4.safety sequence
5.Bankers algorithm
6.exit
2

the number of available :
1520
the number of allocation :
0       0       1       2
1       0       0       0
1       3       5       4
0       6       3       2
0       0       1       4

the number of MAX :
0       0       1       2
1       7       5       0
2       3       5       6
0       6       5       2
0       6       5       6
```



```
1.accept
2.display
3.Need
4.safety sequence
5.Bankers algorithm
6.exit
4

safe sequence is :
P0      P2      P1      P3      P4
1.accept
2.display
3.Need
4.safety sequence
5.Bankers algorithm
6.exit
5
```

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

Slip 4

Slip 5

Q.1Consider 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>
int
available[20],Need[20][20],MAX[20][20],alloction[20][20],work[20],safe[20],Request[20],finish[20],n
,m;

void accept_matrix(int arr[20][20])
{
	int i,j;
	for(i=0;i<n;i++)
	{
		for(j=0;j<m;j++)
		{
			scanf("%d",&arr[i][j]);
		}
	}
}
void accept_array(int arr[20],int no)
{
	int i;
	for(i=0;i<no;i++)
	{

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

	}
}
void display_matrix(int arr[20][20])
{
	int i,j;
	for(i=0;i<n;i++)
	{
		for(j=0;j<m;j++)
		{
			printf("%d\t", arr[i][j]);
		}
		printf("\n");
	}
}
void display_array(int arr[20],int no)
{
	int i;
	for(i=0;i<no;i++)
	{
```

```
                printf("%d", arr[i]);

        }
}
void find_need()
{
        int i,j;
        for(i=0;i<n;i++)
        {
                for(j=0;j<m;j++)
                {
                        Need[i][j]=MAX[i][j] - alloction[i][j];
                }
        }
}
int compare_need(int p)
{
        int i,j,flag=0;
        for(j=0;j<m;j++)
        {
                if(Need[p][j]>work[j])
                {
                        flag=1;
                        break;
                }
        }
        if(flag==0)
        {
                return p;
        }
        return -1;
}
void safety_algo()
{
        int over=0,i,j,k,pno,l=0,flag;
        for(i=0;i<m;i++)
        work[i]=available[i];
        for(i=0;i<n;i++)
        finish[i]=0;
        while(!over)
        {
                        for(i=0;i<n;i++)
                        {
                                if(finish[i]==0)
                                {
                                        flag=0;
                                        pno=compare_need(i);
                                        if(pno>-1)
                                        break;
                                }
                        }
```

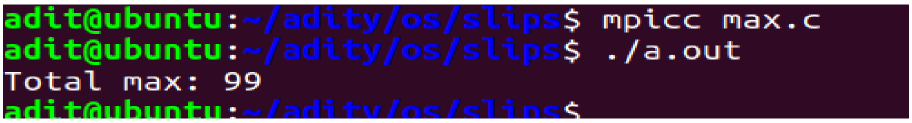
```
            if(i==n)
            {
                printf("system is not safe\n");
                exit(1);
            }
            if(i<n && pno>=0)
            {
                for(k=0;k<m;k++)
                work[k]+=alloction[pno][k];
                finish[pno]=1;
                safe[l++]=pno;
                if(l>=n)
                {
                    printf("\nsafe sequence is :\n");
                    for(l=0;l<n;l++)
                    {
                        printf("P%d\t",safe[l]);
                        over=1;
                    }
                }
            }
        }
}

void bankers_algo(int pno)
{
    int i;
    for(i=0;i<n;i++)
    {
        if(Request[i] > Need[pno][i])
        {
            printf("\nError...process exceeds its Max demand\n");
            return;
        }
    }
    for(i=0;i<n;i++)
    {
        if(Request[i] > available[i])
        {
            printf("\nProcess must wait! Resources not available\n");
            return;
        }
    }
    for(i=0;i<n;i++)
    {
        available[i]=available[i]-Request[i];
        available[i]=available[i]+Request[i];
        Need[pno][i]=Need[pno][i]-Request[i];
    }
    safety_algo();
}
```

```
void main()
{
	printf("\nEnter the number of processes :\n");
	scanf("%d",&n);
	printf("\nEnter the number of Resources :\n");
	scanf("%d",&m);
	int ch;
	while(ch!=6)
	{


		printf("\n1.accept\n2.display\n3.Need\n4.safety sequence\n5.Bankers
algorithm\n6.exit\n");
		scanf("%d", &ch);

		switch(ch)
		{

		case 1:
			printf("\nenter the number of available :\n");
			accept_array(available,m);
			printf("\nenter the number of allocation :\n");
			accept_matrix(alloction);
			printf("\nenter the number of MAx :\n");
			accept_matrix(MAX);
			break;
		case 2:

			printf("\nthe number of available :\n");
			display_array(available,m);
			printf("\nthe number of allocation :\n");
			display_matrix(alloction);
			printf("\nthe number of MAX :\n");
			display_matrix(MAX);
			break;
		case 3:
			find_need();
			printf("\nNeed matrix:\n");
			display_matrix(Need);
			break;
		case 4:
			safety_algo();
			break;

		case 5:
			int a;
			printf("\nEnter the process number\n");
			scanf("%d",&a);
			printf("\nEnter request\n");
			accept_array(Request,m);
```

```
            bankers_algo(a);
            break;

        case 6:
            printf("exit\n");
            break;
        default:
            printf("invalid choice\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)

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define ARRAY_SIZE 1000
int main(int argc, char *argv[]) {
  int rank, size;
  int partial_max = 0;
  int total_max = 0;
  int numbers[ARRAY_SIZE];

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  // Generate random numbers on each process
  srand(rank);
  for (int i = 0; i < ARRAY_SIZE; i++) {
    numbers[i] = rand() % 100;
  }

  // Compute the partial maximum
  for (int i = 0; i < ARRAY_SIZE; i++) {
    if (numbers[i] > partial_max) {
      partial_max = numbers[i];
    }
  }

  // Reduce the partial maximums to find the total maximum
  MPI_Reduce(&partial_max, &total_max, 1, MPI_INT, MPI_MAX, 0, MPI_COMM_WORLD);

  // Print the result on the root process
  if (rank == 0) {
    printf("Total max: %d\n", total_max);
  }

  MPI_Finalize();
  return 0;
}
```



```
adit@ubuntu:~/adity/os/slips$ mpicc max.c
adit@ubuntu:~/adity/os/slips$ ./a.out
Total max: 99
adit@ubuntu:~/adity/os/slips$
```

Slip 7

Q.1 Consider the following snapshot of the system.

Process	Allocation				M a x				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 stateor not

Same as 5(1)

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

Slip 11

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
same as slip 1

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

#define ARRAY_SIZE 1000

int main(int argc, char *argv[]) {
  int rank, size;
  int partial_min = 0;
  int total_min = 0;
  int numbers[ARRAY_SIZE];

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  // Generate random numbers on each process
  srand(rank);
  for (int i = 0; i < ARRAY_SIZE; i++) {
    numbers[i] = rand() % 100;
  }

  // Compute the partial maximum
  for (int i = 0; i < ARRAY_SIZE; i++) {
    if (numbers[i] < partial_min) {
      partial_min = numbers[i];
    }
  }

  // Reduce the partial maximums to find the total maximum
  MPI_Reduce(&partial_min, &total_min, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);

  // Print the result on the root process
  if (rank == 0) {
    printf("Total min: %d\n", total_min);
  }
  MPI_Finalize();
  return 0;
}
```

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.

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

Proces s	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			

Same as slip 3

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

Slip 16

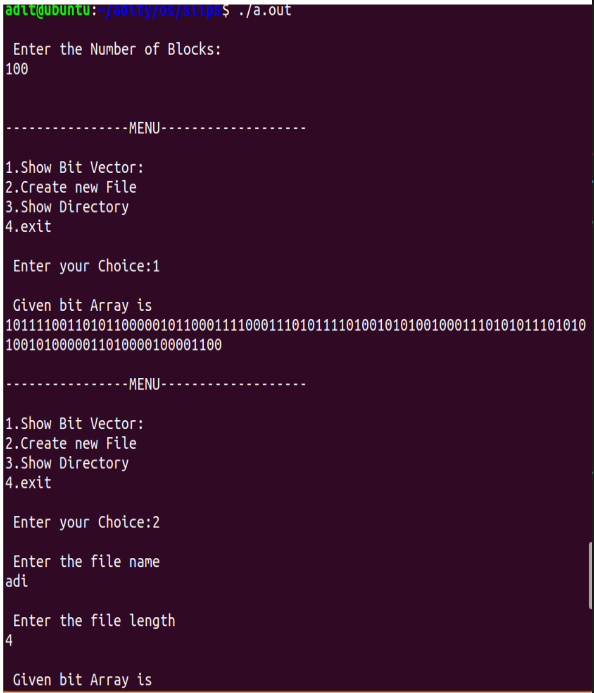
Q.1Write a program to simulate Sequential (Contiguous) file allocation method. Assume disk with n number of blocks. Give value of n as input. Randomly mark some block as allocated and accordingly maintain the list of free blocks Write menu driver program with menu options as mentioned below and implement each option

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

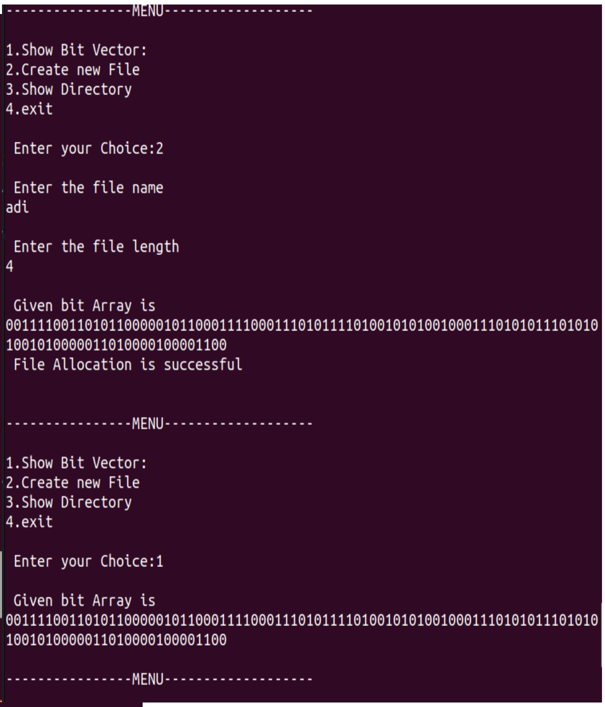
```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>
//#include <conio.h>
struct dir
{
	char filename[20];
	int start;
	int len;
	struct dir *next;
};
struct dir *d,*newNode,*lastNode,*temp;
int Bit[100],n;

void generate_bitVector()
{
	int i,num;
	for(i=0;i<n;i++)
	{
		num=rand();
		Bit[i]=num%2;
	}
}
```

```
void show_bitVector()
{
        int i;
        printf("\n Given bit Array is\n");
        for(i=0;i<=n;i++)
        {
                printf("%d",Bit[i]);
        }
}

void showDirectory()
{
        printf("\n Filename\t Start\t Length\n");
        temp=d;
        while(temp)
        {
        printf("%s\t\t%d\t%d\n",temp->filename,temp->start,temp->len);
        temp=temp->next;
        }
}

void createFile()
{
        char filename[20];
        int i,l,j,k,n1,start,flag;
        printf("\n Enter the file name\n");
        scanf("%s",filename);
        printf("\n Enter the file length\n");
        scanf("%d",&l);
        for(i=0;i<=n;i++)
        {
                if(Bit[i]==1)
                {
                        start=i;
                        flag=1;
                        k=i;
                        for(j=0;j<1;j++)
                        {
                                if(Bit[k]==0)
                                {
                                        flag=0;
                                        start=-1;
                                        break;
                                }
                                k++;
                                }
                        }
                        if(flag==1)
                        {
                                break;
                }
            }
            if(start>=0)
            {
                i=start;
                n1=0;
                while(n1<1)
                {
                    Bit[i]=0;
                    i++;
                    n1++;
                }
                show_bitVector();
                newNode=(struct dir*)malloc(sizeof(struct dir));
                strcpy(newNode->filename,filename);
                newNode->len=1;
                newNode->start=start;
                if(d==NULL)
                {
                    lastNode=d=newNode;
                }
                else
                {
                    lastNode->next=newNode;
                    lastNode=lastNode->next;
                }
                printf("\n File Allocation is successful\n");
            }
            else
            {
                printf("\n File Allocation Failed\n");
            }
    }

int main()
{
    int Bit[100],i,j,k,start,flag,ch;
    d=NULL;
    printf("\n Enter the Number of Blocks:\n");
    scanf("%d",&n);
    generate_bitVector();
    do
    {
        printf("\n \n----------------MENU-------------------");
        printf("\n\n1.Show Bit Vector:\n2.Create new File\n3.Show Directory\n4.exit");
        printf("\n\n Enter your Choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
            show_bitVector();
```

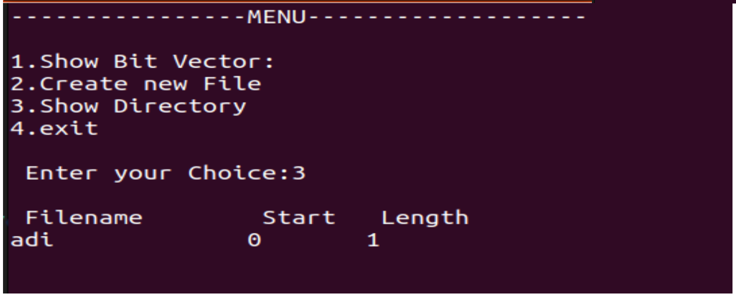
```
                    break;

                    case 2:
                    createFile(d);
                    break;

                    case 3:
                    showDirectory(d);
                    break;


            }
        }while(ch!=5);
        return 0;
}
```



```
adit@ubuntu:~/adity/os/slips$ ./a.out

 Enter the Number of Blocks:
100


---------------MENU------------------

1.Show Bit Vector:
2.Create new File
3.Show Directory
4.exit

 Enter your Choice:1

 Given bit Array is
101111001101011000001011000111100011101011110100101010010001110101011101010
1001010000011010000100001100

---------------MENU------------------

1.Show Bit Vector:
2.Create new File
3.Show Directory
4.exit

 Enter your Choice:2

 Enter the file name
adi

 Enter the file length
4

 Given bit Array is
```



```
---------------MENU------------------

1.Show Bit Vector:
2.Create new File
3.Show Directory
4.exit

 Enter your Choice:2

 Enter the file name
adi

 Enter the file length
4

 Given bit Array is
001111001101011000001011000111100011101011110100101010010001110101011101010
1001010000011010000100001100
 File Allocation is successful


---------------MENU------------------

1.Show Bit Vector:
2.Create new File
3.Show Directory
4.exit

 Enter your Choice:1

 Given bit Array is
001111001101011000001011000111100011101011110100101010010001110101011101010
1001010000011010000100001100

---------------MENU------------------
```



```
---------------MENU------------------

1.Show Bit Vector:
2.Create new File
3.Show Directory
4.exit

 Enter your Choice:3

 Filename         Start    Length
adi               0        1
```

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)

Same as 11

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

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

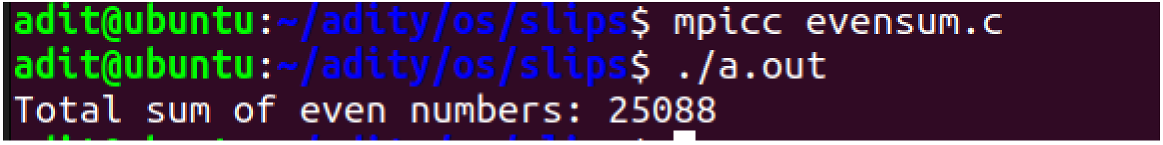
2.Write an MPI program to find the max number from randomly generated 1000 numbers (stored in array) on a cluster (Hint: Use MPI_Reduce)
Same as slip 5

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
Same as slip 1

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

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

#define ARRAY_SIZE 1000

int main(int argc, char *argv[]) {
    int rank, size;
    int partial_sum = 0;
    int total_sum = 0;
    int numbers[ARRAY_SIZE];

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    // Generate random numbers on each process
    srand(rank);
    for (int i = 0; i < ARRAY_SIZE; i++) {
        numbers[i] = rand() % 100;
    }

    // Compute the partial sum of even numbers
    for (int i = 0; i < ARRAY_SIZE; i++) {
        if (numbers[i] % 2 == 0) {
            partial_sum += numbers[i];
        }
    }

    // Reduce the partial sums to calculate the total sum
    MPI_Reduce(&partial_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // Print the result on the root process
    if (rank == 0) {
        printf("Total sum of even numbers: %d\n", total_sum);
    }

    MPI_Finalize();
    return 0;
}
```

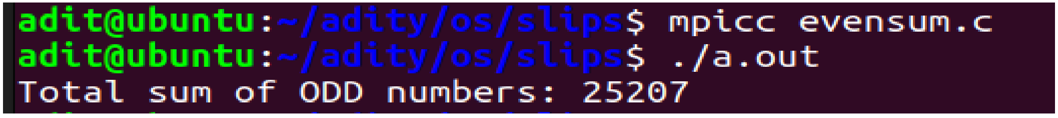


```
adit@ubuntu:~/adity/os/slips$ mpicc evensum.c
adit@ubuntu:~/adity/os/slips$ ./a.out
Total sum of even numbers: 25088
```

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

#define ARRAY_SIZE 1000

int main(int argc, char *argv[]) {
    int rank, size;
    int partial_sum = 0;
    int total_sum = 0;
    int numbers[ARRAY_SIZE];

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Generate random numbers on each process
    srand(rank);
    for (int i = 0; i < ARRAY_SIZE; i++) {
        numbers[i] = rand() % 100;
    }

    // Compute the partial sum of odd numbers
    for (int i = 0; i < ARRAY_SIZE; i++) {
        if (numbers[i] % 2 != 0) {
            partial_sum += numbers[i];
        }
    }

    // Reduce the partial sums to calculate the total sum
    MPI_Reduce(&partial_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // Print the result on the root process
    if (rank == 0) {
        printf("Total sum of even numbers: %d\n", total_sum);
    }

    MPI_Finalize();
    return 0;
}
```



```
adit@ubuntu:~/adity/os/slips$ mpicc evensum.c
adit@ubuntu:~/adity/os/slips$ ./a.out
Total sum of ODD numbers: 25207
```

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

Slip 24

Q.1 Write an MPI program to calculate sum of all odd randomly generated 1000 numbers (stored in array) on a cluster.
Same as slip 22

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.

Proces s	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			

Calculate and display the content of need matrix?

Is the system in safe state? If display the safe sequence.

Same as slip 3

Slip 26

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.

Proces s	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?

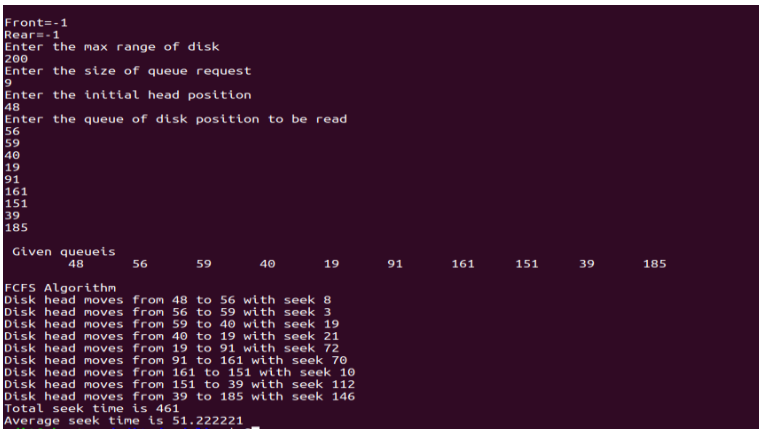
b) Is the system in safe state? If display the safe sequence.

Same as slip 3

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



```
Front=-1
Rear=-1
Enter the max range of disk
200
Enter the size of queue request
9
Enter the initial head position
48
Enter the queue of disk position to be read
56
59
40
19
91
161
151
39
185

 Given queueis
       48      56      59      40      19      91      161     151     39      185

FCFS Algorithm
Disk head moves from 48 to 56 with seek 8
Disk head moves from 56 to 59 with seek 3
Disk head moves from 59 to 40 with seek 19
Disk head moves from 40 to 19 with seek 21
Disk head moves from 19 to 91 with seek 72
Disk head moves from 91 to 161 with seek 70
Disk head moves from 161 to 151 with seek 10
Disk head moves from 151 to 39 with seek 112
Disk head moves from 39 to 185 with seek 146
Total seek time is 461
Average seek time is 51.222221
```

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)

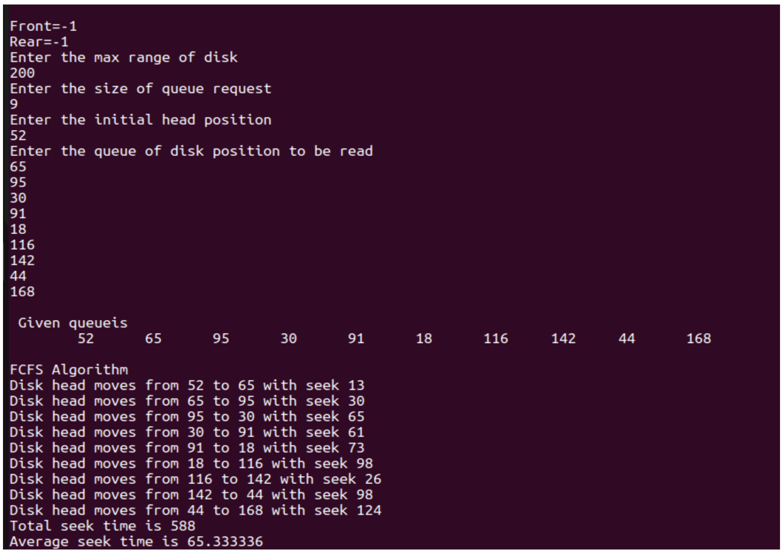
Same as slip 11 .2

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

Same as slip 1



```
Front=-1
Rear=-1
Enter the max range of disk
200
Enter the size of queue request
9
Enter the initial head position
52
Enter the queue of disk position to be read
65
95
30
91
18
116
142
44
168

 Given queueis
        52      65      95      30      91      18      116     142     44      168

FCFS Algorithm
Disk head moves from 52 to 65 with seek 13
Disk head moves from 65 to 95 with seek 30
Disk head moves from 95 to 30 with seek 65
Disk head moves from 30 to 91 with seek 61
Disk head moves from 91 to 18 with seek 73
Disk head moves from 18 to 116 with seek 98
Disk head moves from 116 to 142 with seek 26
Disk head moves from 142 to 44 with seek 98
Disk head moves from 44 to 168 with seek 124
Total seek time is 588
Average seek time is 65.333336
```