slip 1

bankers

```
#include<stdio.h>
# define true 1
# define false 0
int m,n,max[10][10],alloc[10][10],avl[10],need[10][10],finish[10],i,j;

void computeneed()
{
        for(i=0;i<m;i++)
                for(j=0;j<n;j++)
                        need[i][j]=max[i][j]-alloc[i][j];
}
int isfeasible(int pno)
{
        int cnt=0;
        for(j=0;j<n;j++)
                if(need[pno][j]<=avl[j])
                        cnt++;
        if(cnt==n)
                return 1;
        else
                return 0;
}
void checksystem()
{
        int ans[m],cnt=0,flag;
        for(i=0;i<m;i++)
        finish[i]=false;
        while(true)
        {
                flag=false;
                for(i=0;i<m;i++)
                        if(!finish[i])
                        {
                                printf("\n trying for p%d",i);
                                if(isfeasible(i))
                                {
                                        flag=true;
                                        printf("\n process p%d granted
resources\n",i);
                                        finish[i]=true;
                                        ans[cnt++]=i;
                                        for(j=0;j<n;j++)
                                                avl[j]=avl[j]+alloc[i][j];
                                }
                                else
                                        printf("\nprocess p%d cannot be granted
```

```
resources\n",i);
                        }
                if(flag==false)
                break;
        }
        flag=true;
        for(i=0;i<m;i++)
                if(finish[i]==0)
                        flag=false;
        if(flag==1)
        {
                printf("\nSystem is in safe state\n");
                printf("\nSafe sequence is as follows\n");
                for(i=0;i<cnt;i++)
                        printf("p%d\t",ans[i]);
        }
        else
                printf("\nSystem is not in safe state\n");
}

void acceptdata(int x[10][10])
{
        int i,j;
        for(i=0;i<m;i++)
        {
                printf("p%d\n",i);
                for(j=0;j<n;j++)
                {
                        printf("%c:",65+j);
                        scanf("%d",&x[i][j]);
                }
        }
}
void acceptavailability()
{
        int i;
        for(i=0;i<n;i++)
        {
                printf("%c:",65+i);
                scanf("%d",&avl[i]);
        }
}
void displaydata()
{
        int i,j;
        printf("\n\tallocation\t\tmax\t\tneed\n");
        printf("\t");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
```

```
                        printf("%4c",65+j);
                printf("\t");
        }
        for(i=0;i<m;i++)
        {
                printf("\n p%d\t",i);
                for(j=0;j<n;j++)
                        printf("%4d",alloc[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",max[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",need[i][j]);
        }
        printf("\n available\n");
        for(j=0;j<n;j++)
                printf("%4d",avl[j]);
}
int main()
{
        printf("\n enter the no. of processes and resources");
        scanf("%d %d",&m,&n);
        printf("\n enter the allocation\n");
        acceptdata(alloc);
        printf("\n enter the max limit\n");
        acceptdata(max);
        printf("\n enter the availability\n");
        acceptavailability();
        computeneed();
        displaydata();
        checksystem();
}
```

---

fcfs

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i,n,req[50],mov=0,cp;
    printf("enter the current position\n");
    scanf("%d",&cp);
    printf("enter the number of requests\n");
    scanf("%d",&n);
    printf("enter the request order\n");
    for(i=0;i<n;i++)
    {
```

```
        scanf("%d",&req[i]);
    }
    mov=mov+abs(cp-req[0]);
    printf("%d -> %d",cp,req[0]);
    for(i=1;i<n;i++)
    {
        mov=mov+abs(req[i]-req[i-1]);
        printf(" -> %d",req[i]);
    }
    printf("\n");
    printf("total head movement = %d\n",mov);
}
```

****

slip 2

linked file

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

struct block {
         int blkno;
         struct block *next;
};

struct dirfile {char fname[20];
        int length;
        struct block *startblk;
}direntry[20];

int bv[64];
int used=0;
int totalfile=0;
int n;

void initialize()
 {
        int i;
        srand(time(NULL));
        for(i=0;i<n;i++)
        {
                if(rand()%2==0)
                {
                        bv[i]=0;
                        used++;
                }
                else
                {
```

```
                        bv[i]=1;
                }
        }
}
void showbv(){
        int i;
        printf("block number\t status\n");
        for(i=0;i<n;i++){
                printf("%d\t\t",i);
                if(bv[i]==0){
                        printf("allocated\n");
                }else
                {
                        printf("Free\n");
                }
        }
}
int findFreeBlock() {
        for (int i = 0; i < n; ++i) {
                 if (bv[i] == 1) {
                         return i;
                 }
         }
         return -1; // No free block found
}
struct block* allocateBlocks(int length) {
        struct block* start = NULL;
         struct block* current = NULL;
         int allocatedblk=0;
         int blocknum;
         while (allocatedblk < length) {
                 blocknum = findFreeBlock();
                 if (blocknum == -1)
                {
                         printf("Error: No free space available!\n");
                         return NULL;
                 }
        // Allocate block
        bv[blocknum] = 0;
        // Create block node
         struct block* newblock = (struct block*)malloc(sizeof(structblock));
         if (newblock == NULL) {
                 printf("Memory allocation failed!\n");
                 return NULL;
         }
         newblock->blkno = blocknum;
         newblock->next = NULL;
         // Link block to file
         if (start == NULL) {
                 start = newblock;
```

```
         } else {
                 current->next = newblock;
         }
         current = newblock;
         allocatedblk++;
         }
        return start;
 }

void createfile()
{
        char fname[10];
        int length,blknum,k;
        struct block * sblock=NULL;
        printf("\nEnter File Name : ");
        scanf("%s",&fname);
        printf("enter the length of file:");
        scanf("%d",&length);
        sblock = allocateBlocks(length);

         if (sblock == NULL) {
                 printf("File creation failed!\n");
                 return;
         }
        printf("\n block allocated\n");
        used=used+length;
        k=totalfile++;
        strcpy(direntry[k].fname,fname);
        direntry[k].startblk = sblock;
}
void displaydir()
{
        int k;
        printf("\t filename\t start_block\n");
        for(k=0;k<totalfile;k++)
        {
                printf("%s",direntry[k].fname);
                printf("\tBlocks: ");

                 struct block* current = direntry[k].startblk;
                 while (current != NULL) {
                         printf("%d ", current->blkno);
                         current = current->next;
                 }
                 printf("\n\n");
        }
        printf("\n used block=%d",used);
        printf("\n free block =%d\n",n-used);
}
int main()
```

```
{
        int choice;
        printf("enter the number of blocks in the disk:");
        scanf("%d",&n);
        initialize();
        do{
                printf("\n menu:\n");
                printf("1.bit vector \n");
                printf("2.create new file\n");
                printf("3.show directory\n");
                printf("4.exit\n");
                printf("Enter your choice:");
                scanf("%d",&choice);
                switch(choice){
                        case 1:showbv(n);
                         break;
                        case 2:createfile();
                         break;
                        case 3:displaydir();
                         break;
                        case 4:printf("Exiting.....");
                         break;
                        default: printf("Eror:invalid choice\n");
                        break;
                }
        }
        while(choice!=4);
        return 0;
}
```

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

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

#define ARRAY_SIZE 1000

int main(int argc, char* argv[]) {
    int rank, size, i;
    int array[ARRAY_SIZE];
    int local_sum = 0, total_sum;

    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```
    // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Seed the random number generator to get different results each time
    srand(rank + time(NULL));

    // Generate random numbers in each process
    for(i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 100;
        local_sum += array[i];
    }

    // Print the local sum of each process
    printf("Local sum for process %d is %d\n", rank, local_sum);

    // Reduce all of the local sums into the total sum
    MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // Print the total sum once at the root
    if (rank == 0) {
        printf("Total sum = %d\n", total_sum);
    }

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
**********************************************************************
slip 3

bankers

#include<stdio.h>
# define true 1
# define false 0
int m,n,max[10][10],alloc[10][10],avl[10],need[10][10],finish[10],i,j;

void computeneed()
{
        for(i=0;i<m;i++)
                for(j=0;j<n;j++)
                        need[i][j]=max[i][j]-alloc[i][j];
}
int isfeasible(int pno)
{
        int cnt=0;
        for(j=0;j<n;j++)
                if(need[pno][j]<=avl[j])
```

```
                        cnt++;
        if(cnt==n)
                return 1;
        else
                return 0;
}
void checksystem()
{
        int ans[m],cnt=0,flag;
        for(i=0;i<m;i++)
        finish[i]=false;
        while(true)
        {
                flag=false;
                for(i=0;i<m;i++)
                        if(!finish[i])
                        {
                                printf("\n trying for p%d",i);
                                if(isfeasible(i))
                                {
                                        flag=true;
                                        printf("\n process p%d granted
resources\n",i);
                                        finish[i]=true;
                                        ans[cnt++]=i;
                                        for(j=0;j<n;j++)
                                                avl[j]=avl[j]+alloc[i][j];
                                }
                                else
                                        printf("\nprocess p%d cannot be granted
resources\n",i);
                        }
                if(flag==false)
                break;
        }
        flag=true;
        for(i=0;i<m;i++)
                if(finish[i]==0)
                        flag=false;
        if(flag==1)
        {
                printf("\nSystem is in safe state\n");
                printf("\nSafe sequence is as follows\n");
                for(i=0;i<cnt;i++)
                        printf("p%d\t",ans[i]);
        }
        else
                printf("\nSystem is not in safe state\n");
}
```

```
void acceptdata(int x[10][10])
{
        int i,j;
        for(i=0;i<m;i++)
        {
                printf("p%d\n",i);
                for(j=0;j<n;j++)
                {
                        printf("%c:",65+j);
                        scanf("%d",&x[i][j]);
                }
        }
}
void acceptavailability()
{
        int i;
        for(i=0;i<n;i++)
        {
                printf("%c:",65+i);
                scanf("%d",&avl[i]);
        }
}
void displaydata()
{
        int i,j;
        printf("\n\tallocation\t\tmax\t\tneed\n");
        printf("\t");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                        printf("%4c",65+j);
                printf("\t");
        }
        for(i=0;i<m;i++)
        {
                printf("\n p%d\t",i);
                for(j=0;j<n;j++)
                        printf("%4d",alloc[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",max[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",need[i][j]);
        }
        printf("\n available\n");
        for(j=0;j<n;j++)
                printf("%4d",avl[j]);
}
int main()
```

```
{
        printf("\n enter the no. of processes and resources");
        scanf("%d %d",&m,&n);
        printf("\n enter the allocation\n");
        acceptdata(alloc);
        printf("\n enter the max limit\n");
        acceptdata(max);
        printf("\n enter the availability\n");
        acceptavailability();
        computeneed();
        displaydata();
        checksystem();
}
```

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

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

#define ARRAY_SIZE 1000

int main(int argc, char* argv[]) {
    int rank, size, i;
    int array[ARRAY_SIZE];
    int local_sum = 0, total_sum;
    float average;

    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Seed the random number generator to get different results each time
    srand(rank + time(NULL));

    // Generate random numbers in each process
    for(i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 100;
        local_sum += array[i];
    }

    // Print the local sum of each process
    printf("Local sum for process %d is %d\n", rank, local_sum);
```

```
    // Reduce all of the local sums into the total sum
    MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // Calculate the average
    average = total_sum / (float)(ARRAY_SIZE * size);

    // Print the total sum and average once at the root
    if (rank == 0) {
        printf("Total sum = %d\n", total_sum);
        printf("Average = %.2f\n", average);
    }

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
```

****

slip 4

bankers

```
#include<stdio.h>
# define true 1
# define false 0
int m,n,max[10][10],alloc[10][10],avl[10],need[10][10],finish[10],i,j;

void computeneed()
{
        for(i=0;i<m;i++)
                for(j=0;j<n;j++)
                        need[i][j]=max[i][j]-alloc[i][j];
}
int isfeasible(int pno)
{
        int cnt=0;
        for(j=0;j<n;j++)
                if(need[pno][j]<=avl[j])
                        cnt++;
        if(cnt==n)
                return 1;
        else
                return 0;
}
void checksystem()
{
        int ans[m],cnt=0,flag;
        for(i=0;i<m;i++)
        finish[i]=false;
```

```
        while(true)
        {
                flag=false;
                for(i=0;i<m;i++)
                        if(!finish[i])
                        {
                                printf("\n trying for p%d",i);
                                if(isfeasible(i))
                                {
                                        flag=true;
                                        printf("\n process p%d granted
resources\n",i);
                                        finish[i]=true;
                                        ans[cnt++]=i;
                                        for(j=0;j<n;j++)
                                                avl[j]=avl[j]+alloc[i][j];
                                }
                                else
                                        printf("\nprocess p%d cannot be granted
resources\n",i);
                        }
                if(flag==false)
                break;
        }
        flag=true;
        for(i=0;i<m;i++)
                if(finish[i]==0)
                        flag=false;
        if(flag==1)
        {
                printf("\nSystem is in safe state\n");
                printf("\nSafe sequence is as follows\n");
                for(i=0;i<cnt;i++)
                        printf("p%d\t",ans[i]);
        }
        else
                printf("\nSystem is not in safe state\n");
}

void acceptdata(int x[10][10])
{
        int i,j;
        for(i=0;i<m;i++)
        {
                printf("p%d\n",i);
                for(j=0;j<n;j++)
                {
                        printf("%c:",65+j);
                        scanf("%d",&x[i][j]);
                }
```

```
        }
}
void acceptavailability()
{
        int i;
        for(i=0;i<n;i++)
        {
                printf("%c:",65+i);
                scanf("%d",&avl[i]);
        }
}
void displaydata()
{
        int i,j;
        printf("\n\tallocation\t\tmax\t\tneed\n");
        printf("\t");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                        printf("%4c",65+j);
                printf("\t");
        }
        for(i=0;i<m;i++)
        {
                printf("\n p%d\t",i);
                for(j=0;j<n;j++)
                        printf("%4d",alloc[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",max[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",need[i][j]);
        }
        printf("\n available\n");
        for(j=0;j<n;j++)
                printf("%4d",avl[j]);
}
int main()
{
        printf("\n enter the no. of processes and resources");
        scanf("%d %d",&m,&n);
        printf("\n enter the allocation\n");
        acceptdata(alloc);
        printf("\n enter the max limit\n");
        acceptdata(max);
        printf("\n enter the availability\n");
        acceptavailability();
        computeneed();
        displaydata();
```

```
        checksystem();
}
---------------------------------

scan

#include<stdio.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
%d\n",queue[j],queue[j+1],diff);
                }
                printf("Total seek time is %d\n",seek);
                avg=seek/(float)n;
                printf("Average seek time is %f\n",avg);
                return 0;
}
```

****

slip 5

bankers

```
#include<stdio.h>
# define true 1
# define false 0
int m,n,max[10][10],alloc[10][10],avl[10],need[10][10],finish[10],i,j;

void computeneed()
{
        for(i=0;i<m;i++)
                for(j=0;j<n;j++)
                        need[i][j]=max[i][j]-alloc[i][j];
}
int isfeasible(int pno)
{
        int cnt=0;
        for(j=0;j<n;j++)
                if(need[pno][j]<=avl[j])
                        cnt++;
        if(cnt==n)
                return 1;
```

```
        else
                return 0;
}
void checksystem()
{
        int ans[m],cnt=0,flag;
        for(i=0;i<m;i++)
        finish[i]=false;
        while(true)
        {
                flag=false;
                for(i=0;i<m;i++)
                        if(!finish[i])
                        {
                                printf("\n trying for p%d",i);
                                if(isfeasible(i))
                                {
                                        flag=true;
                                        printf("\n process p%d granted
resources\n",i);
                                        finish[i]=true;
                                        ans[cnt++]=i;
                                        for(j=0;j<n;j++)
                                                avl[j]=avl[j]+alloc[i][j];
                                }
                                else
                                        printf("\nprocess p%d cannot be granted
resources\n",i);
                        }
                if(flag==false)
                break;
        }
        flag=true;
        for(i=0;i<m;i++)
                if(finish[i]==0)
                        flag=false;
        if(flag==1)
        {
                printf("\nSystem is in safe state\n");
                printf("\nSafe sequence is as follows\n");
                for(i=0;i<cnt;i++)
                        printf("p%d\t",ans[i]);
        }
        else
                printf("\nSystem is not in safe state\n");
}

void acceptdata(int x[10][10])
{
        int i,j;
```

```
        for(i=0;i<m;i++)
        {
                printf("p%d\n",i);
                for(j=0;j<n;j++)
                {
                        printf("%c:",65+j);
                        scanf("%d",&x[i][j]);
                }
        }
}
void acceptavailability()
{
        int i;
        for(i=0;i<n;i++)
        {
                printf("%c:",65+i);
                scanf("%d",&avl[i]);
        }
}
void displaydata()
{
        int i,j;
        printf("\n\tallocation\t\tmax\t\tneed\n");
        printf("\t");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                        printf("%4c",65+j);
                printf("\t");
        }
        for(i=0;i<m;i++)
        {
                printf("\n p%d\t",i);
                for(j=0;j<n;j++)
                        printf("%4d",alloc[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",max[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",need[i][j]);
        }
        printf("\n available\n");
        for(j=0;j<n;j++)
                printf("%4d",avl[j]);
}
int main()
{
        printf("\n enter the no. of processes and resources");
        scanf("%d %d",&m,&n);
```

```
        printf("\n enter the allocation\n");
        acceptdata(alloc);
        printf("\n enter the max limit\n");
        acceptdata(max);
        printf("\n enter the availability\n");
        acceptavailability();
        computeneed();
        displaydata();
        checksystem();
}
```

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>

#define ARRAY_SIZE 1000

int main(int argc, char* argv[]) {
    int rank, size, i;
    int array[ARRAY_SIZE];
    int local_max, global_max;

    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Seed the random number generator to get different results each time
    srand(rank + time(NULL));

    // Generate random numbers in each process
    for(i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 100;
        if(i == 0 || array[i] > local_max) {
            local_max = array[i];
        }
    }

    // Print the local max of each process
    printf("Local max for process %d is %d\n", rank, local_max);

    // Reduce all of the local maxima into the global max
    MPI_Reduce(&local_max, &global_max, 1, MPI_INT, MPI_MAX, 0, MPI_COMM_WORLD);
```

```
    // Print the global max once at the root
    if (rank == 0) {
        printf("Global max = %d\n", global_max);
    }

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
```

****

slip 6

linked file

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

struct block {
         int blkno;
         struct block *next;
};

struct dirfile {char fname[20];
        int length;
        struct block *startblk;
}direntry[20];

int bv[64];
int used=0;
int totalfile=0;
int n;

void initialize()
 {
        int i;
        srand(time(NULL));
        for(i=0;i<n;i++)
        {
                if(rand()%2==0)
                {
                        bv[i]=0;
                        used++;
                }
                else
                {
                        bv[i]=1;
```

```
                }
        }
}
void showbv(){
        int i;
        printf("block number\t status\n");
        for(i=0;i<n;i++){
                printf("%d\t\t",i);
                if(bv[i]==0){
                        printf("allocated\n");
                }else
                {
                        printf("Free\n");
                }
        }
}
int findFreeBlock() {
        for (int i = 0; i < n; ++i) {
                 if (bv[i] == 1) {
                         return i;
                 }
         }
         return -1; // No free block found
}
struct block* allocateBlocks(int length) {
        struct block* start = NULL;
         struct block* current = NULL;
         int allocatedblk=0;
         int blocknum;
         while (allocatedblk < length) {
                 blocknum = findFreeBlock();
                 if (blocknum == -1)
                {
                         printf("Error: No free space available!\n");
                         return NULL;
                 }
        // Allocate block
        bv[blocknum] = 0;
        // Create block node
         struct block* newblock = (struct block*)malloc(sizeof(structblock));
         if (newblock == NULL) {
                 printf("Memory allocation failed!\n");
                 return NULL;
         }
         newblock->blkno = blocknum;
         newblock->next = NULL;
         // Link block to file
         if (start == NULL) {
                 start = newblock;
         } else {
```

```
                current->next = newblock;
         }
         current = newblock;
         allocatedblk++;
         }
        return start;
 }

void createfile()
{
        char fname[10];
        int length,blknum,k;
        struct block * sblock=NULL;
        printf("\nEnter File Name : ");
        scanf("%s",&fname);
        printf("enter the length of file:");
        scanf("%d",&length);
        sblock = allocateBlocks(length);

         if (sblock == NULL) {
                 printf("File creation failed!\n");
                 return;
         }
        printf("\n block allocated\n");
        used=used+length;
        k=totalfile++;
        strcpy(direntry[k].fname,fname);
        direntry[k].startblk = sblock;
}
void displaydir()
{
        int k;
        printf("\t filename\t start_block\n");
        for(k=0;k<totalfile;k++)
        {
                printf("%s",direntry[k].fname);
                printf("\tBlocks: ");

                 struct block* current = direntry[k].startblk;
                 while (current != NULL) {
                         printf("%d ", current->blkno);
                         current = current->next;
                 }
                 printf("\n\n");
        }
        printf("\n used block=%d",used);
        printf("\n free block =%d\n",n-used);
}
int main()
{
```

```
        int choice;
        printf("enter the number of blocks in the disk:");
        scanf("%d",&n);
        initialize();
        do{
                printf("\n menu:\n");
                printf("1.bit vector \n");
                printf("2.create new file\n");
                printf("3.show directory\n");
                printf("4.exit\n");
                printf("Enter your choice:");
                scanf("%d",&choice);
                switch(choice){
                        case 1:showbv(n);
                         break;
                        case 2:createfile();
                         break;
                        case 3:displaydir();
                         break;
                        case 4:printf("Exiting.....");
                         break;
                        default: printf("Eror:invalid choice\n");
                        break;
                }
        }
        while(choice!=4);
        return 0;
}
-----------------------------------

c scan

#include<stdio.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;
                queue[i+1]=0;
                for(i=temp1+3,j=0;j<temp2;i++,j++)
                queue[i]=queue2[j];
                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
%d\n",queue[j],queue[j+1],diff);
                }
                printf("Total seek time is %d\n",seek);
                avg=seek/(float)n;
                printf("Average seek time is %f\n",avg);
```

```
            return 0;
}
**********************************************************************
slip 7

bankers

#include<stdio.h>
# define true 1
# define false 0
int m,n,max[10][10],alloc[10][10],avl[10],need[10][10],finish[10],i,j;

void computeneed()
{
        for(i=0;i<m;i++)
                for(j=0;j<n;j++)
                        need[i][j]=max[i][j]-alloc[i][j];
}
int isfeasible(int pno)
{
        int cnt=0;
        for(j=0;j<n;j++)
                if(need[pno][j]<=avl[j])
                        cnt++;
        if(cnt==n)
                return 1;
        else
                return 0;
}
void checksystem()
{
        int ans[m],cnt=0,flag;
        for(i=0;i<m;i++)
        finish[i]=false;
        while(true)
        {
                flag=false;
                for(i=0;i<m;i++)
                        if(!finish[i])
                        {
                                printf("\n trying for p%d",i);
                                if(isfeasible(i))
                                {
                                        flag=true;
                                        printf("\n process p%d granted
resources\n",i);
                                        finish[i]=true;
                                        ans[cnt++]=i;
                                        for(j=0;j<n;j++)
                                                avl[j]=avl[j]+alloc[i][j];
```

```
                                }
                                else
                                        printf("\nprocess p%d cannot be granted
resources\n",i);
                        }
                if(flag==false)
                break;
        }
        flag=true;
        for(i=0;i<m;i++)
                if(finish[i]==0)
                        flag=false;
        if(flag==1)
        {
                printf("\nSystem is in safe state\n");
                printf("\nSafe sequence is as follows\n");
                for(i=0;i<cnt;i++)
                        printf("p%d\t",ans[i]);
        }
        else
                printf("\nSystem is not in safe state\n");
}

void acceptdata(int x[10][10])
{
        int i,j;
        for(i=0;i<m;i++)
        {
                printf("p%d\n",i);
                for(j=0;j<n;j++)
                {
                        printf("%c:",65+j);
                        scanf("%d",&x[i][j]);
                }
        }
}
void acceptavailability()
{
        int i;
        for(i=0;i<n;i++)
        {
                printf("%c:",65+i);
                scanf("%d",&avl[i]);
        }
}
void displaydata()
{
        int i,j;
        printf("\n\tallocation\t\tmax\t\tneed\n");
        printf("\t");
```

```
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                        printf("%4c",65+j);
                printf("\t");
        }
        for(i=0;i<m;i++)
        {
                printf("\n p%d\t",i);
                for(j=0;j<n;j++)
                        printf("%4d",alloc[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",max[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",need[i][j]);
        }
        printf("\n available\n");
        for(j=0;j<n;j++)
                printf("%4d",avl[j]);
}
int main()
{
        printf("\n enter the no. of processes and resources");
        scanf("%d %d",&m,&n);
        printf("\n enter the allocation\n");
        acceptdata(alloc);
        printf("\n enter the max limit\n");
        acceptdata(max);
        printf("\n enter the availability\n");
        acceptavailability();
        computeneed();
        displaydata();
        checksystem();
}
```

---

scan

```
#include<stdio.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
%d\n",queue[j],queue[j+1],diff);
            }
            printf("Total seek time is %d\n",seek);
            avg=seek/(float)n;
            printf("Average seek time is %f\n",avg);
            return 0;
}
**********************************************************************
slip 8

contigous allocation

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
struct dirfile
{
        char fname[20];
        int startblk,length;
}direntry[20];

int bv[64];
int used=0;
int totalfile=0;
int n;

void initialize()
{
        int i;
        srand(time(NULL));
        for(i=0;i<n;i++)
        {
                if(rand()%2==0)
                {
                        bv[i]=0;
                        used++;
                }
                else
                {
                        bv[i]=1;
                }
        }
}
void showbv()
{
        int i;
```

```
        printf("block number \t status\n");
        for(i=0;i<n;i++)
        {
                printf("%d\t\t",i);
                if(bv[i]==0)
                {
                        printf("allocated\n");
                }
                else
                {
                        printf("free\n");
                }
        }
}

int search(int length)
{
        int i,j,flag=1,blknum;
        for(i=0;i<n;i++)
        {
                if(bv[i]==1)
                {
                        flag=1;
                        for(blknum=i,j=0;j<length;j++)
                        {
                                if(bv[blknum]==1)
                                {
                                        blknum++;
                                        continue;
                                }
                                else
                                {
                                        flag=0;
                                        break;
                                }
                        }
                        if(flag==1)
                                return i;
                }
        }
        return -1;
}
void createfile()
{
        char fname[10];
        int length,blknum,k;
        printf("\n enter file name:");
        scanf("%s",&fname);
        printf("\n enter the length of file:");
        scanf("%d",&length);
```

```
        if(length<=n-used)
        {
                blknum=search(length);
        }
        else
        {
                blknum=-1;
        }
        if(blknum==-1)
        {
                printf("error:no disk space available\n");
        }
        else
        {
                printf("\nblock allocated\n");
                used=used+length;
                for(k=blknum;k<(blknum+length);k++)
                {
                        bv[k]=0;
                }
                k=totalfile++;
                strcpy(direntry[k].fname,fname);
                direntry[k].startblk=blknum;
                direntry[k].length=length;
        }
}
void displaydir()
{
        int k;
        printf("\tfilename\tstart\tsize\n");
        for(k=0;k<totalfile;k++)
        {

printf("%s\t%d\t%d\n",direntry[k].fname,direntry[k].startblk,direntry[k].length);
        }
        printf("\nused block=%d",used);
        printf("\nfree block=%d",n-used);
}
int main()
{
        int choice;
        printf("enter the number of blocks in the disk:");
        scanf("%d",&n);
        initialize();
        do{
                printf("\nmenu\n");
                printf("1.bit vector\n");
                printf("2.create new file\n");
                printf("3.show directory\n");
                printf("4.exit\n");
```

```
                printf("enter your choice: ");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:showbv(n);
                         break;
                        case 2:createfile(n);
                         break;
                        case 3:displaydir();
                         break;
                        case 4:printf("exiting...\n");
                         break;
                        default:printf("error:invalid choice\n");
                        break;
                }
        }
        while(choice!=4);
        return 0;
}
```

---

sstf

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i,n,k,req[50],mov=0,cp,index[50],min,a[50],j=0,mini,cp1;
    printf("enter the current position\n");
    scanf("%d",&cp);
    printf("enter the number of requests\n");
    scanf("%d",&n);
    cp1=cp;
    printf("enter the request order\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&req[i]);
    }
    for(k=0;k<n;k++)
    {
    for(i=0;i<n;i++)
    {
        index[i]=abs(cp-req[i]);
    }
        min=index[0];
    mini=0;
    for(i=1;i<n;i++)
    {
```

```
        if(min>index[i])
        {
            min=index[i];
            mini=i;
        }
    }
    a[j]=req[mini];
    j++;
    cp=req[mini];
    req[mini]=999;
    }
    printf("Sequence is : ");
    printf("%d",cp1);
    mov=mov+abs(cp1-a[0]);
    printf(" -> %d",a[0]);
    for(i=1;i<n;i++)
    {
        mov=mov+abs(a[i]-a[i-1]);
        printf(" -> %d",a[i]);
    }
    printf("\n");
    printf("total head movement = %d\n",mov);
}
```

****

slip 9

bankers

```
#include<stdio.h>
# define true 1
# define false 0
int m,n,max[10][10],alloc[10][10],avl[10],need[10][10],finish[10],i,j;

void computeneed()
{
        for(i=0;i<m;i++)
                for(j=0;j<n;j++)
                        need[i][j]=max[i][j]-alloc[i][j];
}
int isfeasible(int pno)
{
        int cnt=0;
        for(j=0;j<n;j++)
                if(need[pno][j]<=avl[j])
                        cnt++;
        if(cnt==n)
                return 1;
        else
                return 0;
}
```

```
void checksystem()
{
        int ans[m],cnt=0,flag;
        for(i=0;i<m;i++)
        finish[i]=false;
        while(true)
        {
                flag=false;
                for(i=0;i<m;i++)
                        if(!finish[i])
                        {
                                printf("\n trying for p%d",i);
                                if(isfeasible(i))
                                {
                                        flag=true;
                                        printf("\n process p%d granted
resources\n",i);
                                        finish[i]=true;
                                        ans[cnt++]=i;
                                        for(j=0;j<n;j++)
                                                avl[j]=avl[j]+alloc[i][j];
                                }
                                else
                                        printf("\nprocess p%d cannot be granted
resources\n",i);
                        }
                if(flag==false)
                break;
        }
        flag=true;
        for(i=0;i<m;i++)
                if(finish[i]==0)
                        flag=false;
        if(flag==1)
        {
                printf("\nSystem is in safe state\n");
                printf("\nSafe sequence is as follows\n");
                for(i=0;i<cnt;i++)
                        printf("p%d\t",ans[i]);
        }
        else
                printf("\nSystem is not in safe state\n");
}

void acceptdata(int x[10][10])
{
        int i,j;
        for(i=0;i<m;i++)
        {
                printf("p%d\n",i);
```

```
                for(j=0;j<n;j++)
                {
                        printf("%c:",65+j);
                        scanf("%d",&x[i][j]);
                }
        }
}
void acceptavailability()
{
        int i;
        for(i=0;i<n;i++)
        {
                printf("%c:",65+i);
                scanf("%d",&avl[i]);
        }
}
void displaydata()
{
        int i,j;
        printf("\n\tallocation\t\tmax\t\tneed\n");
        printf("\t");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                        printf("%4c",65+j);
                printf("\t");
        }
        for(i=0;i<m;i++)
        {
                printf("\n p%d\t",i);
                for(j=0;j<n;j++)
                        printf("%4d",alloc[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",max[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",need[i][j]);
        }
        printf("\n available\n");
        for(j=0;j<n;j++)
                printf("%4d",avl[j]);
}
int main()
{
        printf("\n enter the no. of processes and resources");
        scanf("%d %d",&m,&n);
        printf("\n enter the allocation\n");
        acceptdata(alloc);
        printf("\n enter the max limit\n");
```

```
        acceptdata(max);
        printf("\n enter the availability\n");
        acceptavailability();
        computeneed();
        displaydata();
        checksystem();
}
---------------------------------
```

look

```
#include<stdio.h>

void main() {
    int queue[20], n, head, i, j, k, seek = 0, max, diff, temp, queue1[20],
queue2[20], temp1 = 0, temp2 = 0;
    printf("Enter the max range of disk: ");
    scanf("%d", &max);
    printf("Enter the initial head position: ");
    scanf("%d", &head);
    printf("Enter the number of queue elements: ");
    scanf("%d", &n);
    printf("Enter the queue elements: ");
    for(i=1; i<=n; i++) {
        scanf("%d", &temp);
        // Process the queue elements into two separate queues
        if(temp >= head) {
            queue1[temp1] = temp;
            temp1++;
        } else {
            queue2[temp2] = temp;
            temp2++;
        }
    }
    // Sort queue1 - increasing order
    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;
            }
        }
    }
    // Sort queue2 - decreasing order
    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;
            }
        }
    }
    // Join the two queues
    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;
    // Calculate the head movements
    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);
}
```

****

slip 10

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

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

#define ARRAY_SIZE 1000

int main(int argc, char* argv[]) {
    int rank, size, i;
    int array[ARRAY_SIZE];
    int local_sum = 0, total_sum;
    float average;

    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
    // Seed the random number generator to get different results each time
    srand(rank + time(NULL));

    // Generate random numbers in each process
    for(i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 100;
        local_sum += array[i];
    }

    // Print the local sum of each process
    printf("Local sum for process %d is %d\n", rank, local_sum);

    // Reduce all of the local sums into the total sum
    MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // Calculate the average
    average = total_sum / (float)(ARRAY_SIZE * size);

    // Print the total sum and average once at the root
    if (rank == 0) {
        printf("Total sum = %d\n", total_sum);
        printf("Average = %.2f\n", average);
    }

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
-----------------------------------

c scan

#include<stdio.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;
            queue[i+1]=0;
            for(i=temp1+3,j=0;j<temp2;i++,j++)
            queue[i]=queue2[j];
            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
%d\n",queue[j],queue[j+1],diff);
            }
            printf("Total seek time is %d\n",seek);
            avg=seek/(float)n;
```

```
            printf("Average seek time is %f\n",avg);
            return 0;
}
**********************************************************************
slip 11

bankers

#include<stdio.h>
# define true 1
# define false 0
int m,n,max[10][10],alloc[10][10],avl[10],need[10][10],finish[10],i,j;

void computeneed()
{
        for(i=0;i<m;i++)
                for(j=0;j<n;j++)
                        need[i][j]=max[i][j]-alloc[i][j];
}
int isfeasible(int pno)
{
        int cnt=0;
        for(j=0;j<n;j++)
                if(need[pno][j]<=avl[j])
                        cnt++;
        if(cnt==n)
                return 1;
        else
                return 0;
}
void checksystem()
{
        int ans[m],cnt=0,flag;
        for(i=0;i<m;i++)
        finish[i]=false;
        while(true)
        {
                flag=false;
                for(i=0;i<m;i++)
                        if(!finish[i])
                        {
                                printf("\n trying for p%d",i);
                                if(isfeasible(i))
                                {
                                        flag=true;
                                        printf("\n process p%d granted
resources\n",i);
                                        finish[i]=true;
                                        ans[cnt++]=i;
                                        for(j=0;j<n;j++)
```

```
                                                avl[j]=avl[j]+alloc[i][j];
                                }
                                else
                                        printf("\nprocess p%d cannot be granted
resources\n",i);
                        }
                if(flag==false)
                break;
        }
        flag=true;
        for(i=0;i<m;i++)
                if(finish[i]==0)
                        flag=false;
        if(flag==1)
        {
                printf("\nSystem is in safe state\n");
                printf("\nSafe sequence is as follows\n");
                for(i=0;i<cnt;i++)
                        printf("p%d\t",ans[i]);
        }
        else
                printf("\nSystem is not in safe state\n");
}

void acceptdata(int x[10][10])
{
        int i,j;
        for(i=0;i<m;i++)
        {
                printf("p%d\n",i);
                for(j=0;j<n;j++)
                {
                        printf("%c:",65+j);
                        scanf("%d",&x[i][j]);
                }
        }
}
void acceptavailability()
{
        int i;
        for(i=0;i<n;i++)
        {
                printf("%c:",65+i);
                scanf("%d",&avl[i]);
        }
}
void displaydata()
{
        int i,j;
        printf("\n\tallocation\t\tmax\t\tneed\n");
```

```
        printf("\t");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                        printf("%4c",65+j);
                printf("\t");
        }
        for(i=0;i<m;i++)
        {
                printf("\n p%d\t",i);
                for(j=0;j<n;j++)
                        printf("%4d",alloc[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",max[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",need[i][j]);
        }
        printf("\n available\n");
        for(j=0;j<n;j++)
                printf("%4d",avl[j]);
}
int main()
{
        printf("\n enter the no. of processes and resources");
        scanf("%d %d",&m,&n);
        printf("\n enter the allocation\n");
        acceptdata(alloc);
        printf("\n enter the max limit\n");
        acceptdata(max);
        printf("\n enter the availability\n");
        acceptavailability();
        computeneed();
        displaydata();
        checksystem();
}
```

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>

#define ARRAY_SIZE 1000

int main(int argc, char* argv[]) {
    int rank, size, i;
```

```
    int array[ARRAY_SIZE];
    int local_min, global_min;

    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Seed the random number generator to get different results each time
    srand(rank + time(NULL));

    // Generate random numbers in each process
    for(i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 100;
        if(i == 0 || array[i] < local_min) {
            local_min = array[i];
        }
    }

    // Print the local min of each process
    printf("Local min for process %d is %d\n", rank, local_min);

    // Reduce all of the local minima into the global min
    MPI_Reduce(&local_min, &global_min, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);

    // Print the global min once at the root
    if (rank == 0) {
        printf("Global min = %d\n", global_min);
    }

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
```

****

slip 12

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

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

#define ARRAY_SIZE 1000
```

```
int main(int argc, char* argv[]) {
    int rank, size, i;
    int array[ARRAY_SIZE];
    int local_sum = 0, total_sum;
    float average;

    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Seed the random number generator to get different results each time
    srand(rank + time(NULL));

    // Generate random numbers in each process
    for(i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 100;
        local_sum += array[i];
    }

    // Print the local sum of each process
    printf("Local sum for process %d is %d\n", rank, local_sum);

    // Reduce all of the local sums into the total sum
    MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // Calculate the average
    average = total_sum / (float)(ARRAY_SIZE * size);

    // Print the total sum and average once at the root
    if (rank == 0) {
        printf("Total sum = %d\n", total_sum);
        printf("Average = %.2f\n", average);
    }

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
-----------------------------------

c look

#include<stdio.h>
```

```
void main() {
    int queue[20], n, head, i, j, k, seek = 0, max, diff, temp, queue1[20],
queue2[20], temp1 = 0, temp2 = 0;
    printf("Enter the max range of disk: ");
    scanf("%d", &max);
    printf("Enter the initial head position: ");
    scanf("%d", &head);
    printf("Enter the number of queue elements: ");
    scanf("%d", &n);
    printf("Enter the queue elements: ");
    for(i=1; i<=n; i++) {
        scanf("%d", &temp);
        // Process the queue elements into two separate queues
        if(temp >= head) {
            queue1[temp1] = temp;
            temp1++;
        } else {
            queue2[temp2] = temp;
            temp2++;
        }
    }
    // Sort queue1 - increasing order
    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;
            }
        }
    }
    // Sort queue2 - increasing order
    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;
            }
        }
    }
    // Join the two queues
    for(i=1, j=0; j<temp1; i++, j++) {
        queue[i] = queue1[j];
    }
    for(i=temp1+1, j=0; j<temp2; i++, j++) {
        queue[i] = queue2[j];
    }
    // Calculate the head movements
```

```
    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);
}
```

****

slip 13

bankers

```
#include<stdio.h>
# define true 1
# define false 0
int m,n,max[10][10],alloc[10][10],avl[10],need[10][10],finish[10],i,j;

void computeneed()
{
        for(i=0;i<m;i++)
                for(j=0;j<n;j++)
                        need[i][j]=max[i][j]-alloc[i][j];
}
int isfeasible(int pno)
{
        int cnt=0;
        for(j=0;j<n;j++)
                if(need[pno][j]<=avl[j])
                        cnt++;
        if(cnt==n)
                return 1;
        else
                return 0;
}
void checksystem()
{
        int ans[m],cnt=0,flag;
        for(i=0;i<m;i++)
        finish[i]=false;
        while(true)
        {
                flag=false;
                for(i=0;i<m;i++)
                        if(!finish[i])
                        {
                                printf("\n trying for p%d",i);
                                if(isfeasible(i))
                                {
                                        flag=true;
```

```
                                        printf("\n process p%d granted
resources\n",i);
                                        finish[i]=true;
                                        ans[cnt++]=i;
                                        for(j=0;j<n;j++)
                                                avl[j]=avl[j]+alloc[i][j];
                                }
                                else
                                        printf("\nprocess p%d cannot be granted
resources\n",i);
                        }
                if(flag==false)
                break;
        }
        flag=true;
        for(i=0;i<m;i++)
                if(finish[i]==0)
                        flag=false;
        if(flag==1)
        {
                printf("\nSystem is in safe state\n");
                printf("\nSafe sequence is as follows\n");
                for(i=0;i<cnt;i++)
                        printf("p%d\t",ans[i]);
        }
        else
                printf("\nSystem is not in safe state\n");
}

void acceptdata(int x[10][10])
{
        int i,j;
        for(i=0;i<m;i++)
        {
                printf("p%d\n",i);
                for(j=0;j<n;j++)
                {
                        printf("%c:",65+j);
                        scanf("%d",&x[i][j]);
                }
        }
}
void acceptavailability()
{
        int i;
        for(i=0;i<n;i++)
        {
                printf("%c:",65+i);
                scanf("%d",&avl[i]);
        }
```

```
}
void displaydata()
{
        int i,j;
        printf("\n\tallocation\t\tmax\t\tneed\n");
        printf("\t");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                        printf("%4c",65+j);
                printf("\t");
        }
        for(i=0;i<m;i++)
        {
                printf("\n p%d\t",i);
                for(j=0;j<n;j++)
                        printf("%4d",alloc[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",max[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",need[i][j]);
        }
        printf("\n available\n");
        for(j=0;j<n;j++)
                printf("%4d",avl[j]);
}
int main()
{
        printf("\n enter the no. of processes and resources");
        scanf("%d %d",&m,&n);
        printf("\n enter the allocation\n");
        acceptdata(alloc);
        printf("\n enter the max limit\n");
        acceptdata(max);
        printf("\n enter the availability\n");
        acceptavailability();
        computeneed();
        displaydata();
        checksystem();
}
-----------------------------------

scan

#include<stdio.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
%d\n",queue[j],queue[j+1],diff);
            }
            printf("Total seek time is %d\n",seek);
            avg=seek/(float)n;
            printf("Average seek time is %f\n",avg);
            return 0;
}
*********************************************************************
slip 14
contigous allocation

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
struct dirfile
{
        char fname[20];
        int startblk,length;
}direntry[20];

int bv[64];
int used=0;
int totalfile=0;
int n;

void initialize()
{
        int i;
        srand(time(NULL));
        for(i=0;i<n;i++)
        {
                if(rand()%2==0)
                {
                        bv[i]=0;
                        used++;
                }
                else
                {
                        bv[i]=1;
                }
```

```
        }
}
void showbv()
{
        int i;
        printf("block number \t status\n");
        for(i=0;i<n;i++)
        {
                printf("%d\t\t",i);
                if(bv[i]==0)
                {
                        printf("allocated\n");
                }
                else
                {
                        printf("free\n");
                }
        }
}

int search(int length)
{
        int i,j,flag=1,blknum;
        for(i=0;i<n;i++)
        {
                if(bv[i]==1)
                {
                        flag=1;
                        for(blknum=i,j=0;j<length;j++)
                        {
                                if(bv[blknum]==1)
                                {
                                        blknum++;
                                        continue;
                                }
                                else
                                {
                                        flag=0;
                                        break;
                                }
                        }
                        if(flag==1)
                                return i;
                }
        }
        return -1;
}
void createfile()
{
        char fname[10];
```

```
        int length,blknum,k;
        printf("\n enter file name:");
        scanf("%s",&fname);
        printf("\n enter the length of file:");
        scanf("%d",&length);
        if(length<=n-used)
        {
                blknum=search(length);
        }
        else
        {
                blknum=-1;
        }
        if(blknum==-1)
        {
                printf("error:no disk space available\n");
        }
        else
        {
                printf("\nblock allocated\n");
                used=used+length;
                for(k=blknum;k<(blknum+length);k++)
                {
                        bv[k]=0;
                }
                k=totalfile++;
                strcpy(direntry[k].fname,fname);
                direntry[k].startblk=blknum;
                direntry[k].length=length;
        }
}
void displaydir()
{
        int k;
        printf("\tfilename\tstart\tsize\n");
        for(k=0;k<totalfile;k++)
        {

printf("%s\t%d\t%d\n",direntry[k].fname,direntry[k].startblk,direntry[k].length);
        }
        printf("\nused block=%d",used);
        printf("\nfree block=%d",n-used);
}
int main()
{
        int choice;
        printf("enter the number of blocks in the disk:");
        scanf("%d",&n);
        initialize();
        do{
```

```
                printf("\nmenu\n");
                printf("1.bit vector\n");
                printf("2.create new file\n");
                printf("3.show directory\n");
                printf("4.exit\n");
                printf("enter your choice: ");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:showbv(n);
                         break;
                        case 2:createfile(n);
                         break;
                        case 3:displaydir();
                         break;
                        case 4:printf("exiting...\n");
                         break;
                        default:printf("error:invalid choice\n");
                        break;
                }
        }
        while(choice!=4);
        return 0;
}
```

---

sstf

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i,n,k,req[50],mov=0,cp,index[50],min,a[50],j=0,mini,cp1;
    printf("enter the current position\n");
    scanf("%d",&cp);
    printf("enter the number of requests\n");
    scanf("%d",&n);
    cp1=cp;
    printf("enter the request order\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&req[i]);
    }
    for(k=0;k<n;k++)
    {
    for(i=0;i<n;i++)
    {
        index[i]=abs(cp-req[i]);
    }
```

```
        min=index[0];
    mini=0;
    for(i=1;i<n;i++)
    {
        if(min>index[i])
        {
            min=index[i];
            mini=i;
        }
    }
    a[j]=req[mini];
    j++;
    cp=req[mini];
    req[mini]=999;
    }
    printf("Sequence is : ");
    printf("%d",cp1);
    mov=mov+abs(cp1-a[0]);
    printf(" -> %d",a[0]);
    for(i=1;i<n;i++)
    {
        mov=mov+abs(a[i]-a[i-1]);
        printf(" -> %d",a[i]);
    }
    printf("\n");
    printf("total head movement = %d\n",mov);
}
```

****

slip 15

linked file

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

struct block {
         int blkno;
         struct block *next;
};

struct dirfile {char fname[20];
        int length;
        struct block *startblk;
}direntry[20];

int bv[64];
int used=0;
int totalfile=0;
```

```
int n;

void initialize()
 {
        int i;
        srand(time(NULL));
        for(i=0;i<n;i++)
        {
                if(rand()%2==0)
                {
                        bv[i]=0;
                        used++;
                }
                else
                {
                        bv[i]=1;
                }
        }
}
void showbv(){
        int i;
        printf("block number\t status\n");
        for(i=0;i<n;i++){
                printf("%d\t\t",i);
                if(bv[i]==0){
                        printf("allocated\n");
                }else
                {
                        printf("Free\n");
                }
        }
}
int findFreeBlock() {
        for (int i = 0; i < n; ++i) {
                 if (bv[i] == 1) {
                         return i;
                 }
         }
         return -1; // No free block found
}
struct block* allocateBlocks(int length) {
        struct block* start = NULL;
         struct block* current = NULL;
         int allocatedblk=0;
         int blocknum;
         while (allocatedblk < length) {
                 blocknum = findFreeBlock();
                 if (blocknum == -1)
                {
                         printf("Error: No free space available!\n");
```

```
                        return NULL;
                }
        // Allocate block
        bv[blocknum] = 0;
        // Create block node
         struct block* newblock = (struct block*)malloc(sizeof(structblock));
         if (newblock == NULL) {
                printf("Memory allocation failed!\n");
                return NULL;
         }
         newblock->blkno = blocknum;
         newblock->next = NULL;
         // Link block to file
         if (start == NULL) {
                start = newblock;
         } else {
                current->next = newblock;
         }
         current = newblock;
         allocatedblk++;
         }
        return start;
 }

void createfile()
{
        char fname[10];
        int length,blknum,k;
        struct block * sblock=NULL;
        printf("\nEnter File Name : ");
        scanf("%s",&fname);
        printf("enter the length of file:");
        scanf("%d",&length);
        sblock = allocateBlocks(length);

         if (sblock == NULL) {
                printf("File creation failed!\n");
                return;
         }
        printf("\n block allocated\n");
        used=used+length;
        k=totalfile++;
        strcpy(direntry[k].fname,fname);
        direntry[k].startblk = sblock;
}
void displaydir()
{
        int k;
        printf("\t filename\t start_block\n");
        for(k=0;k<totalfile;k++)
```

```
        {
                printf("%s",direntry[k].fname);
                printf("\tBlocks: ");

                 struct block* current = direntry[k].startblk;
                 while (current != NULL) {
                         printf("%d ", current->blkno);
                         current = current->next;
                 }
                 printf("\n\n");
        }
        printf("\n used block=%d",used);
        printf("\n free block =%d\n",n-used);
}
int main()
{
        int choice;
        printf("enter the number of blocks in the disk:");
        scanf("%d",&n);
        initialize();
        do{
                printf("\n menu:\n");
                printf("1.bit vector \n");
                printf("2.create new file\n");
                printf("3.show directory\n");
                printf("4.exit\n");
                printf("Enter your choice:");
                scanf("%d",&choice);
                switch(choice){
                        case 1:showbv(n);
                         break;
                        case 2:createfile();
                         break;
                        case 3:displaydir();
                         break;
                        case 4:printf("Exiting.....");
                         break;
                        default: printf("Eror:invalid choice\n");
                        break;
                }
        }
        while(choice!=4);
        return 0;
}
-----------------------------------

c scan

#include<stdio.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;
            queue[i+1]=0;
            for(i=temp1+3,j=0;j<temp2;i++,j++)
            queue[i]=queue2[j];
            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
%d\n",queue[j],queue[j+1],diff);
            }
            printf("Total seek time is %d\n",seek);
            avg=seek/(float)n;
            printf("Average seek time is %f\n",avg);
            return 0;
}
```

****

slip 16

contigous allocation

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
struct dirfile
{
        char fname[20];
        int startblk,length;
}direntry[20];

int bv[64];
int used=0;
int totalfile=0;
int n;

void initialize()
{
        int i;
        srand(time(NULL));
        for(i=0;i<n;i++)
        {
                if(rand()%2==0)
                {
                        bv[i]=0;
                        used++;
                }
                else
```

```
                {
                        bv[i]=1;
                }
        }
}
void showbv()
{
        int i;
        printf("block number \t status\n");
        for(i=0;i<n;i++)
        {
                printf("%d\t\t",i);
                if(bv[i]==0)
                {
                        printf("allocated\n");
                }
                else
                {
                        printf("free\n");
                }
        }
}

int search(int length)
{
        int i,j,flag=1,blknum;
        for(i=0;i<n;i++)
        {
                if(bv[i]==1)
                {
                        flag=1;
                        for(blknum=i,j=0;j<length;j++)
                        {
                                if(bv[blknum]==1)
                                {
                                        blknum++;
                                        continue;
                                }
                                else
                                {
                                        flag=0;
                                        break;
                                }
                        }
                        if(flag==1)
                                return i;
                }
        }
        return -1;
}
```

```
void createfile()
{
        char fname[10];
        int length,blknum,k;
        printf("\n enter file name:");
        scanf("%s",&fname);
        printf("\n enter the length of file:");
        scanf("%d",&length);
        if(length<=n-used)
        {
                blknum=search(length);
        }
        else
        {
                blknum=-1;
        }
        if(blknum==-1)
        {
                printf("error:no disk space available\n");
        }
        else
        {
                printf("\nblock allocated\n");
                used=used+length;
                for(k=blknum;k<(blknum+length);k++)
                {
                        bv[k]=0;
                }
                k=totalfile++;
                strcpy(direntry[k].fname,fname);
                direntry[k].startblk=blknum;
                direntry[k].length=length;
        }
}
void displaydir()
{
        int k;
        printf("\tfilename\tstart\tsize\n");
        for(k=0;k<totalfile;k++)
        {

printf("%s\t%d\t%d\n",direntry[k].fname,direntry[k].startblk,direntry[k].length);
        }
        printf("\nused block=%d",used);
        printf("\nfree block=%d",n-used);
}
int main()
{
        int choice;
        printf("enter the number of blocks in the disk:");
```

```
        scanf("%d",&n);
        initialize();
        do{
                printf("\nmenu\n");
                printf("1.bit vector\n");
                printf("2.create new file\n");
                printf("3.show directory\n");
                printf("4.exit\n");
                printf("enter your choice: ");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:showbv(n);
                         break;
                        case 2:createfile(n);
                         break;
                        case 3:displaydir();
                         break;
                        case 4:printf("exiting...\n");
                         break;
                        default:printf("error:invalid choice\n");
                        break;
                }
        }
        while(choice!=4);
        return 0;
}
```

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>

#define ARRAY_SIZE 1000

int main(int argc, char* argv[]) {
    int rank, size, i;
    int array[ARRAY_SIZE];
    int local_min, global_min;

    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Get the rank of the process
```

```
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Seed the random number generator to get different results each time
    srand(rank + time(NULL));

    // Generate random numbers in each process
    for(i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 100;
        if(i == 0 || array[i] < local_min) {
            local_min = array[i];
        }
    }

    // Print the local min of each process
    printf("Local min for process %d is %d\n", rank, local_min);

    // Reduce all of the local minima into the global min
    MPI_Reduce(&local_min, &global_min, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);

    // Print the global min once at the root
    if (rank == 0) {
        printf("Global min = %d\n", global_min);
    }

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
**********************************************************************
slip 17

index file all

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>

struct dirfile
{
        char fname[20];
        int length;
        int indexblock;
        int a[10];
}direntry[20];

int bv[64];
int used=0;
int totalfile=0;
```

```
int n;
void initialize()
{
        int i;
        srand(time(NULL));
        for(i=0;i<n;i++){
                if(rand()%2==0){
                        bv[i]=0;
                        used++;
                }else{
                        bv[i]=1;
                }
        }
}
void showbv(){
        int i;
        printf("block number\t status\n");
        for(i=0;i<n;i++){
                printf("%d\t\t",i);
                if(bv[i]==0){
                        printf("allocated\n");
                }else{
                        printf("Free\n");
                }
        }
}
int findFreeBlock() {
        for (int i = 0; i < n; ++i) {
                 if (bv[i] == 1) {
                 return i;
                 }
         }
         return -1; // No free block found
}
void allocateBlocks(int length) {
        int allocatedblk=0;
         int blocknum;
        direntry[totalfile].indexblock=0;
         while (allocatedblk < length) {
                 blocknum = findFreeBlock();
                 if (blocknum == -1) {
                         printf("Error: No free space available!\n");
                        return ;
                 }

                // Allocate block
                bv[blocknum] = 0;
                if (direntry[totalfile].indexblock == 0)
                {
                        direntry[totalfile].indexblock = blocknum;
```

```
                }
                else
                {
                        direntry[totalfile].a[allocatedblk]=blocknum;
                         allocatedblk++;
                        // printf("\nManisha");
                }
         }
 }

void createfile()
{
        char fname[10];
        int length,blknum,k;
        // struct block * sblock=NULL;
        printf("\nEnter File Name : ");
        scanf("%s",&fname);
        printf("enter the length of file:");
        scanf("%d",&length);
        allocateBlocks(length);

         // if (sblock == NULL) {
         // printf("File creation failed!\n");
         // return;
         // }
        printf("\n block allocated\n");
        used=used+length;
        k=totalfile++;
        strcpy(direntry[k].fname,fname);
        direntry[k].length = length;
}
void displaydir()
{
        int k,i=0;
        printf("\t filename\t start_block\n");
        for(k=0;k<totalfile;k++)
        {
                printf("%s",direntry[k].fname);
                printf("Index Block = %d",direntry[k].indexblock);
                printf("\tLength = %d",direntry[k].length);
                printf("\tBlocks: ");
                i=0;
                 while (i < direntry[k].length) {
                         printf("\t%d ",direntry[k].a[i] );
                         i++;
                 }
                 printf("\n\n");
        }
        printf("\n used block=%d",used);
        printf("\n free block =%d\n",n-used);
```

```
}

int main()
{
        int choice;
        printf("enter the number of blocks in the disk:");
        scanf("%d",&n);
        initialize();
        do{
                printf("\n menu:\n");
                printf("1.bit vector \n");
                printf("2.create new file\n");
                printf("3.show directory\n");
                printf("4.exit\n");
                printf("Enter your choice:");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:showbv(n);
                         break;
                        case 2:createfile();
                         break;
                        case 3:displaydir();
                         break;
                        case 4:printf("Exiting.....");
                         break;
                        default: printf("Eror:invalid choice\n");
                        break;
                }
        }while(choice!=4);
        return 0;
}
```

---

look

```
#include<stdio.h>

void main() {
    int queue[20], n, head, i, j, k, seek = 0, max, diff, temp, queue1[20],
queue2[20], temp1 = 0, temp2 = 0;
    printf("Enter the max range of disk: ");
    scanf("%d", &max);
    printf("Enter the initial head position: ");
    scanf("%d", &head);
    printf("Enter the number of queue elements: ");
    scanf("%d", &n);
    printf("Enter the queue elements: ");
    for(i=1; i<=n; i++) {
        scanf("%d", &temp);
```

```
        // Process the queue elements into two separate queues
        if(temp >= head) {
            queue1[temp1] = temp;
            temp1++;
        } else {
            queue2[temp2] = temp;
            temp2++;
        }
    }
    // Sort queue1 - increasing order
    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;
            }
        }
    }
    // Sort queue2 - decreasing order
    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;
            }
        }
    }
    // Join the two queues
    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;
    // Calculate the head movements
    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);
}
```

****

slip 18

index file all

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

struct dirfile
{
        char fname[20];
        int length;
        int indexblock;
        int a[10];
}direntry[20];

int bv[64];
int used=0;
int totalfile=0;
int n;
void initialize()
{
        int i;
        srand(time(NULL));
        for(i=0;i<n;i++){
                if(rand()%2==0){
                        bv[i]=0;
                        used++;
                }else{
                        bv[i]=1;
                }
        }
}
void showbv(){
        int i;
        printf("block number\t status\n");
        for(i=0;i<n;i++){
                printf("%d\t\t",i);
                if(bv[i]==0){
                        printf("allocated\n");
                }else{
                        printf("Free\n");
                }
        }
}
int findFreeBlock() {
        for (int i = 0; i < n; ++i) {
                 if (bv[i] == 1) {
                 return i;
                 }
```

```
         }
         return -1; // No free block found
}
void allocateBlocks(int length) {
        int allocatedblk=0;
         int blocknum;
        direntry[totalfile].indexblock=0;
         while (allocatedblk < length) {
                 blocknum = findFreeBlock();
                 if (blocknum == -1) {
                         printf("Error: No free space available!\n");
                        return ;
                 }

                // Allocate block
                bv[blocknum] = 0;
                if (direntry[totalfile].indexblock == 0)
                {
                        direntry[totalfile].indexblock = blocknum;
                }
                else
                {
                        direntry[totalfile].a[allocatedblk]=blocknum;
                         allocatedblk++;
                        // printf("\nManisha");
                }
         }
 }

void createfile()
{
        char fname[10];
        int length,blknum,k;
        // struct block * sblock=NULL;
        printf("\nEnter File Name : ");
        scanf("%s",&fname);
        printf("enter the length of file:");
        scanf("%d",&length);
        allocateBlocks(length);

         // if (sblock == NULL) {
         // printf("File creation failed!\n");
         // return;
         // }
        printf("\n block allocated\n");
        used=used+length;
        k=totalfile++;
        strcpy(direntry[k].fname,fname);
        direntry[k].length = length;
}
```

```
void displaydir()
{
        int k,i=0;
        printf("\t filename\t start_block\n");
        for(k=0;k<totalfile;k++)
        {
                printf("%s",direntry[k].fname);
                printf("Index Block = %d",direntry[k].indexblock);
                printf("\tLength = %d",direntry[k].length);
                printf("\tBlocks: ");
                i=0;
                 while (i < direntry[k].length) {
                         printf("\t%d ",direntry[k].a[i] );
                         i++;
                 }
                 printf("\n\n");
        }
        printf("\n used block=%d",used);
        printf("\n free block =%d\n",n-used);
}

int main()
{
        int choice;
        printf("enter the number of blocks in the disk:");
        scanf("%d",&n);
        initialize();
        do{
                printf("\n menu:\n");
                printf("1.bit vector \n");
                printf("2.create new file\n");
                printf("3.show directory\n");
                printf("4.exit\n");
                printf("Enter your choice:");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:showbv(n);
                         break;
                        case 2:createfile();
                         break;
                        case 3:displaydir();
                         break;
                        case 4:printf("Exiting.....");
                         break;
                        default: printf("Eror:invalid choice\n");
                        break;
                }
        }while(choice!=4);
        return 0;
```

```
}
-----------------------------------

scan

#include<stdio.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
%d\n",queue[j],queue[j+1],diff);
                        }
                        printf("Total seek time is %d\n",seek);
                        avg=seek/(float)n;
                        printf("Average seek time is %f\n",avg);
                        return 0;
}
*********************************************************************
slip 19

bankers

#include<stdio.h>
# define true 1
# define false 0
int m,n,max[10][10],alloc[10][10],avl[10],need[10][10],finish[10],i,j;

void computeneed()
{
        for(i=0;i<m;i++)
                for(j=0;j<n;j++)
                        need[i][j]=max[i][j]-alloc[i][j];
}
int isfeasible(int pno)
{
        int cnt=0;
        for(j=0;j<n;j++)
                if(need[pno][j]<=avl[j])
                        cnt++;
        if(cnt==n)
                return 1;
        else
```

```
                return 0;
}
void checksystem()
{
        int ans[m],cnt=0,flag;
        for(i=0;i<m;i++)
        finish[i]=false;
        while(true)
        {
                flag=false;
                for(i=0;i<m;i++)
                        if(!finish[i])
                        {
                                printf("\n trying for p%d",i);
                                if(isfeasible(i))
                                {
                                        flag=true;
                                        printf("\n process p%d granted
resources\n",i);
                                        finish[i]=true;
                                        ans[cnt++]=i;
                                        for(j=0;j<n;j++)
                                                avl[j]=avl[j]+alloc[i][j];
                                }
                                else
                                        printf("\nprocess p%d cannot be granted
resources\n",i);
                        }
                if(flag==false)
                break;
        }
        flag=true;
        for(i=0;i<m;i++)
                if(finish[i]==0)
                        flag=false;
        if(flag==1)
        {
                printf("\nSystem is in safe state\n");
                printf("\nSafe sequence is as follows\n");
                for(i=0;i<cnt;i++)
                        printf("p%d\t",ans[i]);
        }
        else
                printf("\nSystem is not in safe state\n");
}

void acceptdata(int x[10][10])
{
        int i,j;
        for(i=0;i<m;i++)
```

```
        {
                printf("p%d\n",i);
                for(j=0;j<n;j++)
                {
                        printf("%c:",65+j);
                        scanf("%d",&x[i][j]);
                }
        }
}
void acceptavailability()
{
        int i;
        for(i=0;i<n;i++)
        {
                printf("%c:",65+i);
                scanf("%d",&avl[i]);
        }
}
void displaydata()
{
        int i,j;
        printf("\n\tallocation\t\tmax\t\tneed\n");
        printf("\t");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                        printf("%4c",65+j);
                printf("\t");
        }
        for(i=0;i<m;i++)
        {
                printf("\n p%d\t",i);
                for(j=0;j<n;j++)
                        printf("%4d",alloc[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",max[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",need[i][j]);
        }
        printf("\n available\n");
        for(j=0;j<n;j++)
                printf("%4d",avl[j]);
}
int main()
{
        printf("\n enter the no. of processes and resources");
        scanf("%d %d",&m,&n);
        printf("\n enter the allocation\n");
```

```
        acceptdata(alloc);
        printf("\n enter the max limit\n");
        acceptdata(max);
        printf("\n enter the availability\n");
        acceptavailability();
        computeneed();
        displaydata();
        checksystem();
}
-----------------------------------

c scan

#include<stdio.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;
                queue[i+1]=0;
                for(i=temp1+3,j=0;j<temp2;i++,j++)
                queue[i]=queue2[j];
                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
%d\n",queue[j],queue[j+1],diff);
                }
                printf("Total seek time is %d\n",seek);
                avg=seek/(float)n;
                printf("Average seek time is %f\n",avg);
                return 0;
}
**********************************************************************
slip 20

scan

#include<stdio.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
%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;
}
```

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>

#define ARRAY_SIZE 1000

int main(int argc, char* argv[]) {
    int rank, size, i;
    int array[ARRAY_SIZE];
    int local_max, global_max;

    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Seed the random number generator to get different results each time
    srand(rank + time(NULL));

    // Generate random numbers in each process
    for(i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 100;
        if(i == 0 || array[i] > local_max) {
            local_max = array[i];
        }
    }

    // Print the local max of each process
    printf("Local max for process %d is %d\n", rank, local_max);

    // Reduce all of the local maxima into the global max
    MPI_Reduce(&local_max, &global_max, 1, MPI_INT, MPI_MAX, 0, MPI_COMM_WORLD);

    // Print the global max once at the root
```

```
    if (rank == 0) {
        printf("Global max = %d\n", global_max);
    }

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
```

****

slip 21

fcfs

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i,n,req[50],mov=0,cp;
    printf("enter the current position\n");
    scanf("%d",&cp);
    printf("enter the number of requests\n");
    scanf("%d",&n);
    printf("enter the request order\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&req[i]);
    }
    mov=mov+abs(cp-req[0]);
    printf("%d -> %d",cp,req[0]);
    for(i=1;i<n;i++)
    {
        mov=mov+abs(req[i]-req[i-1]);
        printf(" -> %d",req[i]);
    }
    printf("\n");
    printf("total head movement = %d\n",mov);
}
```

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>

#define ARRAY_SIZE 1000
```

```
int main(int argc, char* argv[]) {
    int rank, size, i;
    int array[ARRAY_SIZE];
    int local_sum = 0, total_sum;

    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Seed the random number generator to get different results each time
    srand(rank + time(NULL));

    // Generate random numbers in each process and add to local sum if even
    for(i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 100;
        if(array[i] % 2 == 0) {
            local_sum += array[i];
        }
    }

    // Print the local sum of each process
    printf("Local sum for process %d is %d\n", rank, local_sum);

    // Reduce all of the local sums into the total sum
    MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // Print the total sum once at the root
    if (rank == 0) {
        printf("Total sum = %d\n", total_sum);
    }

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
**********************************************************************
slip 22

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>
```

```
#define ARRAY_SIZE 1000

int main(int argc, char* argv[]) {
    int rank, size, i;
    int array[ARRAY_SIZE];
    int local_sum = 0, total_sum;

    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Seed the random number generator to get different results each time
    srand(rank + time(NULL));

    // Generate random numbers in each process and add to local sum if odd
    for(i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 100;
        if(array[i] % 2 != 0) {
            local_sum += array[i];
        }
    }

    // Print the local sum of each process
    printf("Local sum for process %d is %d\n", rank, local_sum);

    // Reduce all of the local sums into the total sum
    MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // Print the total sum once at the root
    if (rank == 0) {
        printf("Total sum = %d\n", total_sum);
    }

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
-----------------------------------

contigous allocation

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

```
#include<string.h>
#include<time.h>
struct dirfile
{
        char fname[20];
        int startblk,length;
}direntry[20];

int bv[64];
int used=0;
int totalfile=0;
int n;

void initialize()
{
        int i;
        srand(time(NULL));
        for(i=0;i<n;i++)
        {
                if(rand()%2==0)
                {
                        bv[i]=0;
                        used++;
                }
                else
                {
                        bv[i]=1;
                }
        }
}
void showbv()
{
        int i;
        printf("block number \t status\n");
        for(i=0;i<n;i++)
        {
                printf("%d\t\t",i);
                if(bv[i]==0)
                {
                        printf("allocated\n");
                }
                else
                {
                        printf("free\n");
                }
        }
}

int search(int length)
{
```

```
        int i,j,flag=1,blknum;
        for(i=0;i<n;i++)
        {
                if(bv[i]==1)
                {
                        flag=1;
                        for(blknum=i,j=0;j<length;j++)
                        {
                                if(bv[blknum]==1)
                                {
                                        blknum++;
                                        continue;
                                }
                                else
                                {
                                        flag=0;
                                        break;
                                }
                        }
                        if(flag==1)
                                return i;
                }
        }
        return -1;
}
void createfile()
{
        char fname[10];
        int length,blknum,k;
        printf("\n enter file name:");
        scanf("%s",&fname);
        printf("\n enter the length of file:");
        scanf("%d",&length);
        if(length<=n-used)
        {
                blknum=search(length);
        }
        else
        {
                blknum=-1;
        }
        if(blknum==-1)
        {
                printf("error:no disk space available\n");
        }
        else
        {
                printf("\nblock allocated\n");
                used=used+length;
                for(k=blknum;k<(blknum+length);k++)
```

```
                {
                        bv[k]=0;
                }
                k=totalfile++;
                strcpy(direntry[k].fname,fname);
                direntry[k].startblk=blknum;
                direntry[k].length=length;
        }
}
void displaydir()
{
        int k;
        printf("\tfilename\tstart\tsize\n");
        for(k=0;k<totalfile;k++)
        {

printf("%s\t%d\t%d\n",direntry[k].fname,direntry[k].startblk,direntry[k].length);
        }
        printf("\nused block=%d",used);
        printf("\nfree block=%d",n-used);
}
int main()
{
        int choice;
        printf("enter the number of blocks in the disk:");
        scanf("%d",&n);
        initialize();
        do{
                printf("\nmenu\n");
                printf("1.bit vector\n");
                printf("2.create new file\n");
                printf("3.show directory\n");
                printf("4.exit\n");
                printf("enter your choice: ");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:showbv(n);
                         break;
                        case 2:createfile(n);
                         break;
                        case 3:displaydir();
                         break;
                        case 4:printf("exiting...\n");
                         break;
                        default:printf("error:invalid choice\n");
                        break;
                }
        }
        while(choice!=4);
```

```
        return 0;
}

**********************************************************************
slip 23

bankers

#include<stdio.h>
# define true 1
# define false 0
int m,n,max[10][10],alloc[10][10],avl[10],need[10][10],finish[10],i,j;

void computeneed()
{
        for(i=0;i<m;i++)
                for(j=0;j<n;j++)
                        need[i][j]=max[i][j]-alloc[i][j];
}
int isfeasible(int pno)
{
        int cnt=0;
        for(j=0;j<n;j++)
                if(need[pno][j]<=avl[j])
                        cnt++;
        if(cnt==n)
                return 1;
        else
                return 0;
}
void checksystem()
{
        int ans[m],cnt=0,flag;
        for(i=0;i<m;i++)
        finish[i]=false;
        while(true)
        {
                flag=false;
                for(i=0;i<m;i++)
                        if(!finish[i])
                        {
                                printf("\n trying for p%d",i);
                                if(isfeasible(i))
                                {
                                        flag=true;
                                        printf("\n process p%d granted
resources\n",i);
                                        finish[i]=true;
                                        ans[cnt++]=i;
                                        for(j=0;j<n;j++)
```

```
                                                avl[j]=avl[j]+alloc[i][j];
                                }
                                else
                                        printf("\nprocess p%d cannot be granted
resources\n",i);
                        }
                if(flag==false)
                break;
        }
        flag=true;
        for(i=0;i<m;i++)
                if(finish[i]==0)
                        flag=false;
        if(flag==1)
        {
                printf("\nSystem is in safe state\n");
                printf("\nSafe sequence is as follows\n");
                for(i=0;i<cnt;i++)
                        printf("p%d\t",ans[i]);
        }
        else
                printf("\nSystem is not in safe state\n");
}

void acceptdata(int x[10][10])
{
        int i,j;
        for(i=0;i<m;i++)
        {
                printf("p%d\n",i);
                for(j=0;j<n;j++)
                {
                        printf("%c:",65+j);
                        scanf("%d",&x[i][j]);
                }
        }
}
void acceptavailability()
{
        int i;
        for(i=0;i<n;i++)
        {
                printf("%c:",65+i);
                scanf("%d",&avl[i]);
        }
}
void displaydata()
{
        int i,j;
        printf("\n\tallocation\t\tmax\t\tneed\n");
```

```
        printf("\t");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                        printf("%4c",65+j);
                printf("\t");
        }
        for(i=0;i<m;i++)
        {
                printf("\n p%d\t",i);
                for(j=0;j<n;j++)
                        printf("%4d",alloc[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",max[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",need[i][j]);
        }
        printf("\n available\n");
        for(j=0;j<n;j++)
                printf("%4d",avl[j]);
}
int main()
{
        printf("\n enter the no. of processes and resources");
        scanf("%d %d",&m,&n);
        printf("\n enter the allocation\n");
        acceptdata(alloc);
        printf("\n enter the max limit\n");
        acceptdata(max);
        printf("\n enter the availability\n");
        acceptavailability();
        computeneed();
        displaydata();
        checksystem();
}
```

---

sstf

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i,n,k,req[50],mov=0,cp,index[50],min,a[50],j=0,mini,cp1;
    printf("enter the current position\n");
    scanf("%d",&cp);
    printf("enter the number of requests\n");
```

```
    scanf("%d",&n);
    cp1=cp;
    printf("enter the request order\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&req[i]);
    }
    for(k=0;k<n;k++)
    {
    for(i=0;i<n;i++)
    {
        index[i]=abs(cp-req[i]);
    }
        min=index[0];
    mini=0;
    for(i=1;i<n;i++)
    {
        if(min>index[i])
        {
            min=index[i];
            mini=i;
        }
    }
    a[j]=req[mini];
    j++;
    cp=req[mini];
    req[mini]=999;
    }
    printf("Sequence is : ");
    printf("%d",cp1);
    mov=mov+abs(cp1-a[0]);
    printf(" -> %d",a[0]);
    for(i=1;i<n;i++)
    {
        mov=mov+abs(a[i]-a[i-1]);
        printf(" -> %d",a[i]);
    }
    printf("\n");
    printf("total head movement = %d\n",mov);
}
```

****

slip 24

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>
```

```
#define ARRAY_SIZE 1000

int main(int argc, char* argv[]) {
    int rank, size, i;
    int array[ARRAY_SIZE];
    int local_sum = 0, total_sum;

    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Seed the random number generator to get different results each time
    srand(rank + time(NULL));

    // Generate random numbers in each process and add to local sum if odd
    for(i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 100;
        if(array[i] % 2 != 0) {
            local_sum += array[i];
        }
    }

    // Print the local sum of each process
    printf("Local sum for process %d is %d\n", rank, local_sum);

    // Reduce all of the local sums into the total sum
    MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // Print the total sum once at the root
    if (rank == 0) {
        printf("Total sum = %d\n", total_sum);
    }

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
```

---

bankers

```
#include<stdio.h>
# define true 1
# define false 0
```

```
int m,n,max[10][10],alloc[10][10],avl[10],need[10][10],finish[10],i,j;

void computeneed()
{
        for(i=0;i<m;i++)
                for(j=0;j<n;j++)
                        need[i][j]=max[i][j]-alloc[i][j];
}
int isfeasible(int pno)
{
        int cnt=0;
        for(j=0;j<n;j++)
                if(need[pno][j]<=avl[j])
                        cnt++;
        if(cnt==n)
                return 1;
        else
                return 0;
}
void checksystem()
{
        int ans[m],cnt=0,flag;
        for(i=0;i<m;i++)
        finish[i]=false;
        while(true)
        {
                flag=false;
                for(i=0;i<m;i++)
                        if(!finish[i])
                        {
                                printf("\n trying for p%d",i);
                                if(isfeasible(i))
                                {
                                        flag=true;
                                        printf("\n process p%d granted
resources\n",i);
                                        finish[i]=true;
                                        ans[cnt++]=i;
                                        for(j=0;j<n;j++)
                                                avl[j]=avl[j]+alloc[i][j];
                                }
                                else
                                        printf("\nprocess p%d cannot be granted
resources\n",i);
                        }
                if(flag==false)
                break;
        }
        flag=true;
        for(i=0;i<m;i++)
```

```
                if(finish[i]==0)
                        flag=false;
        if(flag==1)
        {
                printf("\nSystem is in safe state\n");
                printf("\nSafe sequence is as follows\n");
                for(i=0;i<cnt;i++)
                        printf("p%d\t",ans[i]);
        }
        else
                printf("\nSystem is not in safe state\n");
}

void acceptdata(int x[10][10])
{
        int i,j;
        for(i=0;i<m;i++)
        {
                printf("p%d\n",i);
                for(j=0;j<n;j++)
                {
                        printf("%c:",65+j);
                        scanf("%d",&x[i][j]);
                }
        }
}
void acceptavailability()
{
        int i;
        for(i=0;i<n;i++)
        {
                printf("%c:",65+i);
                scanf("%d",&avl[i]);
        }
}
void displaydata()
{
        int i,j;
        printf("\n\tallocation\t\tmax\t\tneed\n");
        printf("\t");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                        printf("%4c",65+j);
                printf("\t");
        }
        for(i=0;i<m;i++)
        {
                printf("\n p%d\t",i);
                for(j=0;j<n;j++)
```

```
                        printf("%4d",alloc[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",max[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",need[i][j]);
        }
        printf("\n available\n");
        for(j=0;j<n;j++)
                printf("%4d",avl[j]);
}
int main()
{
        printf("\n enter the no. of processes and resources");
        scanf("%d %d",&m,&n);
        printf("\n enter the allocation\n");
        acceptdata(alloc);
        printf("\n enter the max limit\n");
        acceptdata(max);
        printf("\n enter the availability\n");
        acceptavailability();
        computeneed();
        displaydata();
        checksystem();
}
```

****

slip 25

look

```
#include<stdio.h>

void main() {
    int queue[20], n, head, i, j, k, seek = 0, max, diff, temp, queue1[20],
queue2[20], temp1 = 0, temp2 = 0;
    printf("Enter the max range of disk: ");
    scanf("%d", &max);
    printf("Enter the initial head position: ");
    scanf("%d", &head);
    printf("Enter the number of queue elements: ");
    scanf("%d", &n);
    printf("Enter the queue elements: ");
    for(i=1; i<=n; i++) {
        scanf("%d", &temp);
        // Process the queue elements into two separate queues
        if(temp >= head) {
            queue1[temp1] = temp;
            temp1++;
        } else {
```

```
            queue2[temp2] = temp;
            temp2++;
        }
    }
    // Sort queue1 - increasing order
    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;
            }
        }
    }
    // Sort queue2 - decreasing order
    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;
            }
        }
    }
    // Join the two queues
    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;
    // Calculate the head movements
    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);
}
```

---

linked file

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

```
#include<time.h>

struct block {
         int blkno;
         struct block *next;
};

struct dirfile {char fname[20];
        int length;
        struct block *startblk;
}direntry[20];

int bv[64];
int used=0;
int totalfile=0;
int n;

void initialize()
 {
        int i;
        srand(time(NULL));
        for(i=0;i<n;i++)
        {
                if(rand()%2==0)
                {
                        bv[i]=0;
                        used++;
                }
                else
                {
                        bv[i]=1;
                }
        }
}
void showbv(){
        int i;
        printf("block number\t status\n");
        for(i=0;i<n;i++){
                printf("%d\t\t",i);
                if(bv[i]==0){
                        printf("allocated\n");
                }else
                {
                        printf("Free\n");
                }
        }
}
int findFreeBlock() {
        for (int i = 0; i < n; ++i) {
                 if (bv[i] == 1) {
```

```
                        return i;
                }
        }
        return -1; // No free block found
}
struct block* allocateBlocks(int length) {
        struct block* start = NULL;
        struct block* current = NULL;
        int allocatedblk=0;
        int blocknum;
        while (allocatedblk < length) {
                blocknum = findFreeBlock();
                if (blocknum == -1)
               {
                        printf("Error: No free space available!\n");
                        return NULL;
                }
        // Allocate block
        bv[blocknum] = 0;
        // Create block node
        struct block* newblock = (struct block*)malloc(sizeof(structblock));
        if (newblock == NULL) {
                printf("Memory allocation failed!\n");
                return NULL;
        }
        newblock->blkno = blocknum;
        newblock->next = NULL;
        // Link block to file
        if (start == NULL) {
                start = newblock;
        } else {
                current->next = newblock;
        }
        current = newblock;
        allocatedblk++;
        }
        return start;
 }

void createfile()
{
        char fname[10];
        int length,blknum,k;
        struct block * sblock=NULL;
        printf("\nEnter File Name : ");
        scanf("%s",&fname);
        printf("enter the length of file:");
        scanf("%d",&length);
        sblock = allocateBlocks(length);
```

```
         if (sblock == NULL) {
                 printf("File creation failed!\n");
                 return;
         }
        printf("\n block allocated\n");
        used=used+length;
        k=totalfile++;
        strcpy(direntry[k].fname,fname);
        direntry[k].startblk = sblock;
}
void displaydir()
{
        int k;
        printf("\t filename\t start_block\n");
        for(k=0;k<totalfile;k++)
        {
                printf("%s",direntry[k].fname);
                printf("\tBlocks: ");

                 struct block* current = direntry[k].startblk;
                 while (current != NULL) {
                         printf("%d ", current->blkno);
                         current = current->next;
                 }
                 printf("\n\n");
        }
        printf("\n used block=%d",used);
        printf("\n free block =%d\n",n-used);
}
int main()
{
        int choice;
        printf("enter the number of blocks in the disk:");
        scanf("%d",&n);
        initialize();
        do{
                printf("\n menu:\n");
                printf("1.bit vector \n");
                printf("2.create new file\n");
                printf("3.show directory\n");
                printf("4.exit\n");
                printf("Enter your choice:");
                scanf("%d",&choice);
                switch(choice){
                        case 1:showbv(n);
                         break;
                        case 2:createfile();
                         break;
                        case 3:displaydir();
                         break;
```

```
                        case 4:printf("Exiting.....");
                         break;
                        default: printf("Eror:invalid choice\n");
                        break;
                }
        }
        while(choice!=4);
        return 0;
}
```

****

slip 26

bankers

```
#include<stdio.h>
# define true 1
# define false 0
int m,n,max[10][10],alloc[10][10],avl[10],need[10][10],finish[10],i,j;

void computeneed()
{
        for(i=0;i<m;i++)
                for(j=0;j<n;j++)
                        need[i][j]=max[i][j]-alloc[i][j];
}
int isfeasible(int pno)
{
        int cnt=0;
        for(j=0;j<n;j++)
                if(need[pno][j]<=avl[j])
                        cnt++;
        if(cnt==n)
                return 1;
        else
                return 0;
}
void checksystem()
{
        int ans[m],cnt=0,flag;
        for(i=0;i<m;i++)
        finish[i]=false;
        while(true)
        {
                flag=false;
                for(i=0;i<m;i++)
                        if(!finish[i])
                        {
                                printf("\n trying for p%d",i);
                                if(isfeasible(i))
                                {
```

```
                                        flag=true;
                                        printf("\n process p%d granted
resources\n",i);
                                        finish[i]=true;
                                        ans[cnt++]=i;
                                        for(j=0;j<n;j++)
                                                avl[j]=avl[j]+alloc[i][j];
                                }
                                else
                                        printf("\nprocess p%d cannot be granted
resources\n",i);
                        }
                if(flag==false)
                break;
        }
        flag=true;
        for(i=0;i<m;i++)
                if(finish[i]==0)
                        flag=false;
        if(flag==1)
        {
                printf("\nSystem is in safe state\n");
                printf("\nSafe sequence is as follows\n");
                for(i=0;i<cnt;i++)
                        printf("p%d\t",ans[i]);
        }
        else
                printf("\nSystem is not in safe state\n");
}

void acceptdata(int x[10][10])
{
        int i,j;
        for(i=0;i<m;i++)
        {
                printf("p%d\n",i);
                for(j=0;j<n;j++)
                {
                        printf("%c:",65+j);
                        scanf("%d",&x[i][j]);
                }
        }
}
void acceptavailability()
{
        int i;
        for(i=0;i<n;i++)
        {
                printf("%c:",65+i);
                scanf("%d",&avl[i]);
```

```
        }
}
void displaydata()
{
        int i,j;
        printf("\n\tallocation\t\tmax\t\tneed\n");
        printf("\t");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                        printf("%4c",65+j);
                printf("\t");
        }
        for(i=0;i<m;i++)
        {
                printf("\n p%d\t",i);
                for(j=0;j<n;j++)
                        printf("%4d",alloc[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",max[i][j]);
                printf("\t");
                for(j=0;j<n;j++)
                        printf("%4d",need[i][j]);
        }
        printf("\n available\n");
        for(j=0;j<n;j++)
                printf("%4d",avl[j]);
}
int main()
{
        printf("\n enter the no. of processes and resources");
        scanf("%d %d",&m,&n);
        printf("\n enter the allocation\n");
        acceptdata(alloc);
        printf("\n enter the max limit\n");
        acceptdata(max);
        printf("\n enter the availability\n");
        acceptavailability();
        computeneed();
        displaydata();
        checksystem();
}
```

---

fcfs

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

```
{
    int i,n,req[50],mov=0,cp;
    printf("enter the current position\n");
    scanf("%d",&cp);
    printf("enter the number of requests\n");
    scanf("%d",&n);
    printf("enter the request order\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&req[i]);
    }
    mov=mov+abs(cp-req[0]);
    printf("%d -> %d",cp,req[0]);
    for(i=1;i<n;i++)
    {
        mov=mov+abs(req[i]-req[i-1]);
        printf(" -> %d",req[i]);
    }
    printf("\n");
    printf("total head movement = %d\n",mov);
}
```

****

slip 27

look

```
#include<stdio.h>

void main() {
    int queue[20], n, head, i, j, k, seek = 0, max, diff, temp, queue1[20],
queue2[20], temp1 = 0, temp2 = 0;
    printf("Enter the max range of disk: ");
    scanf("%d", &max);
    printf("Enter the initial head position: ");
    scanf("%d", &head);
    printf("Enter the number of queue elements: ");
    scanf("%d", &n);
    printf("Enter the queue elements: ");
    for(i=1; i<=n; i++) {
        scanf("%d", &temp);
        // Process the queue elements into two separate queues
        if(temp >= head) {
            queue1[temp1] = temp;
            temp1++;
        } else {
            queue2[temp2] = temp;
            temp2++;
        }
    }
    // Sort queue1 - increasing order
```

```
    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;
            }
        }
    }
    // Sort queue2 - decreasing order
    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;
            }
        }
    }
    // Join the two queues
    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;
    // Calculate the head movements
    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);
}
```

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>

#define ARRAY_SIZE 1000

int main(int argc, char* argv[]) {
    int rank, size, i;
```

```
    int array[ARRAY_SIZE];
    int local_min, global_min;

    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Seed the random number generator to get different results each time
    srand(rank + time(NULL));

    // Generate random numbers in each process
    for(i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 100;
        if(i == 0 || array[i] < local_min) {
            local_min = array[i];
        }
    }

    // Print the local min of each process
    printf("Local min for process %d is %d\n", rank, local_min);

    // Reduce all of the local minima into the global min
    MPI_Reduce(&local_min, &global_min, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);

    // Print the global min once at the root
    if (rank == 0) {
        printf("Global min = %d\n", global_min);
    }

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
**********************************************************************
slip 28

c look

#include<stdio.h>

void main() {
    int queue[20], n, head, i, j, k, seek = 0, max, diff, temp, queue1[20],
queue2[20], temp1 = 0, temp2 = 0;
    printf("Enter the max range of disk: ");
```

```
    scanf("%d", &max);
    printf("Enter the initial head position: ");
    scanf("%d", &head);
    printf("Enter the number of queue elements: ");
    scanf("%d", &n);
    printf("Enter the queue elements: ");
    for(i=1; i<=n; i++) {
        scanf("%d", &temp);
        // Process the queue elements into two separate queues
        if(temp >= head) {
            queue1[temp1] = temp;
            temp1++;
        } else {
            queue2[temp2] = temp;
            temp2++;
        }
    }
    // Sort queue1 - increasing order
    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;
            }
        }
    }
    // Sort queue2 - increasing order
    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;
            }
        }
    }
    // Join the two queues
    for(i=1, j=0; j<temp1; i++, j++) {
        queue[i] = queue1[j];
    }
    for(i=temp1+1, j=0; j<temp2; i++, j++) {
        queue[i] = queue2[j];
    }
    // Calculate the head movements
    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);
}
```

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

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

#define ARRAY_SIZE 1000

int main(int argc, char* argv[]) {
    int rank, size, i;
    int array[ARRAY_SIZE];
    int local_sum = 0, total_sum;

    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Seed the random number generator to get different results each time
    srand(rank + time(NULL));

    // Generate random numbers in each process
    for(i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 100;
        local_sum += array[i];
    }

    // Print the local sum of each process
    printf("Local sum for process %d is %d\n", rank, local_sum);

    // Reduce all of the local sums into the total sum
    MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // Print the total sum once at the root
    if (rank == 0) {
        printf("Total sum = %d\n", total_sum);
    }

    // Finalize the MPI environment
    MPI_Finalize();
```

```
    return 0;
}
**********************************************************************
slip 29

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>

#define ARRAY_SIZE 1000

int main(int argc, char* argv[]) {
    int rank, size, i;
    int array[ARRAY_SIZE];
    int local_sum = 0, total_sum;

    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Seed the random number generator to get different results each time
    srand(rank + time(NULL));

    // Generate random numbers in each process and add to local sum if even
    for(i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 100;
        if(array[i] % 2 == 0) {
            local_sum += array[i];
        }
    }

    // Print the local sum of each process
    printf("Local sum for process %d is %d\n", rank, local_sum);

    // Reduce all of the local sums into the total sum
    MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // Print the total sum once at the root
    if (rank == 0) {
        printf("Total sum = %d\n", total_sum);
    }
```

```
    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
```

---

c look

```
#include<stdio.h>

void main() {
    int queue[20], n, head, i, j, k, seek = 0, max, diff, temp, queue1[20],
queue2[20], temp1 = 0, temp2 = 0;
    printf("Enter the max range of disk: ");
    scanf("%d", &max);
    printf("Enter the initial head position: ");
    scanf("%d", &head);
    printf("Enter the number of queue elements: ");
    scanf("%d", &n);
    printf("Enter the queue elements: ");
    for(i=1; i<=n; i++) {
        scanf("%d", &temp);
        // Process the queue elements into two separate queues
        if(temp >= head) {
            queue1[temp1] = temp;
            temp1++;
        } else {
            queue2[temp2] = temp;
            temp2++;
        }
    }
    // Sort queue1 - increasing order
    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;
            }
        }
    }
    // Sort queue2 - increasing order
    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;
```

```
            }
        }
    }
    // Join the two queues
    for(i=1, j=0; j<temp1; i++, j++) {
        queue[i] = queue1[j];
    }
    for(i=temp1+1, j=0; j<temp2; i++, j++) {
        queue[i] = queue2[j];
    }
    // Calculate the head movements
    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);
}
```

****

slip 30

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>

#define ARRAY_SIZE 1000

int main(int argc, char* argv[]) {
    int rank, size, i;
    int array[ARRAY_SIZE];
    int local_min, global_min;

    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Seed the random number generator to get different results each time
    srand(rank + time(NULL));

    // Generate random numbers in each process
    for(i = 0; i < ARRAY_SIZE; i++) {
        array[i] = rand() % 100;
```

```
        if(i == 0 || array[i] < local_min) {
            local_min = array[i];
        }
    }

    // Print the local min of each process
    printf("Local min for process %d is %d\n", rank, local_min);

    // Reduce all of the local minima into the global min
    MPI_Reduce(&local_min, &global_min, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);

    // Print the global min once at the root
    if (rank == 0) {
        printf("Global min = %d\n", global_min);
    }

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
```

---

fcfs

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i,n,req[50],mov=0,cp;
    printf("enter the current position\n");
    scanf("%d",&cp);
    printf("enter the number of requests\n");
    scanf("%d",&n);
    printf("enter the request order\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&req[i]);
    }
    mov=mov+abs(cp-req[0]);
    printf("%d -> %d",cp,req[0]);
    for(i=1;i<n;i++)
    {
        mov=mov+abs(req[i]-req[i-1]);
        printf(" -> %d",req[i]);
    }
    printf("\n");
    printf("total head movement = %d\n",mov);
}
```

****

delete contigous all

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
struct dirfile
{
        char fname[20];
        int startblk,length;
}direntry[20];

int bv[64];
int used=0;
int totalfile=0;
int n;

void initialize()
{
        int i;
        srand(time(NULL));
        for(i=0;i<n;i++)
        {
                if(rand()%2==0)
                {
                        bv[i]=0;
                        used++;
                }
                else
                {
                        bv[i]=1;
                }
        }
}
void showbv()
{
        int i;
        printf("block number \t status\n");
        for(i=0;i<n;i++)
        {
                printf("%d\t\t",i);
                if(bv[i]==0)
                {
                        printf("allocated\n");
                }
                else
                {
                        printf("free\n");
                }
```

```
        }
}
int search(int length)
{
        int i,j,flag=1,blknum;
        for(i=0;i<n;i++)
        {
                if(bv[i]==1)
                {
                        flag=1;
                        for(blknum=i,j=0;j<length;j++)
                        {
                                if(bv[blknum]==1)
                                {
                                        blknum++;
                                        continue;
                                }
                                else
                                {
                                        flag=0;
                                        break;
                                }
                        }
                        if(flag==1)
                        return i;
                }
        }
        return -1;
}
void createfile()
{
        char fname[10];
        int length,blknum,k;
        printf("\n enter file name:");
        scanf("%s",&fname);
        printf("\n enter the length of file:");
         scanf("%d",&length);
        if(length<=n-used)
        {
                blknum=search(length);
        }
        else
        {
                blknum=-1;
        }
        if(blknum==-1)
        {
                printf("error:no disk space available\n");
        }
        else
```

```
        {
                printf("\nblock allocated\n");
                used=used+length;
                for(k=blknum;k<(blknum+length);k++)
                {
                        bv[k]=0;
                }
                k=totalfile++;
                strcpy(direntry[k].fname,fname);
                direntry[k].startblk=blknum;
                direntry[k].length=length;
        }
}
void displaydir()
{
        int k;
        printf("\tfilename\tstart\tsize\n");
        for(k=0;k<totalfile;k++)
        {

printf("\t%s\t\t%d\t%d\n",direntry[k].fname,direntry[k].startblk,direntry[k].length
);
        }
        printf("\nused block=%d",used);
        printf("\nfree block=%d",n-used);
}
void deletefile(char fn[])
{
        int k,i,m,p,flag=1;
        for(k=0;k<totalfile;k++)
        {
                if(strcmp(fn,direntry[k].fname)==0)
                {
                        strcpy(direntry[k].fname,"NULL");
                        m=direntry[k].startblk;
                        p=direntry[k].length;
                        for(i=m;i<(m+p);i++)
                        {
                                bv[i]=1;
                        }
                        flag=0;
                        break;
                }
        }
        if(flag==1)
        printf("\n%s does not exist\n",fn);
        else
                printf("\n%s is deleted\n",fn);
}
int main()
```

```
{
        int choice;
        printf("enter the number of blocks in the disk:");
        scanf("%d",&n);
        initialize();
        do{
                printf("\nmenu\n");
                printf("1.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: ");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:showbv(n);
                         break;
                        case 2:createfile(n);
                         break;
                        case 3:displaydir();
                         break;
                        case 4:char fname[10];
                         printf("\nenter file name:");
                         scanf("%s",&fname);
                         deletefile(fname);
                         break;
                        case 5:printf("exiting...\n");
                         break;
                        default:printf("error:invalid choice\n");
                        break;
                }
        }
        while(choice!=5);
        return 0;
}
```