TY BSC(CS) OS PRACTICAL PROGRAMS
ASSIGNMENT NO.1

Slot I & Slot II

I) Add the following functionalities in your program
a) Accept Available
b) Display Allocation, Max
c) Display the contents of need matrix
d) Display Available

II) Modify above program so as to include the following:
a) Accept Request for a process
b) Resource request algorithm
c) Safety algorithm
Consider a system with 'n' processes and 'm' resource types. Accept number of instances for every resource type. For each process accept the allocation and maximum requirement matrices. Write a program to display the contents of need matrix and to check if the given request of a process can be granted immediately or not

```
#include<stdio.h>

#define MAX 10

int m,n,total[MAX],avail[MAX],alloc[MAX][MAX],

   max[MAX][MAX],need[MAX][MAX],work[MAX],finish[MAX],

   seq[MAX],request[MAX];

void accept()

{

 int i,j;

printf("Enter no.of process:");

 scanf("%d",&n);

 printf("Enter no.of resource types:");

 scanf("%d",&m);

 printf("Enter total no.of resources of each resource type:\n");

 for(i=0;i<m;i++)

 {
```

```
 printf("%c:",65+i);
 scanf("%d",&total[i]);
}
printf("Enter no.of allocated resources of each resource type by each process:\n");
for(i=0;i<n;i++)
{
 printf("P%d:\n",i);
 for(j=0;j<m;j++)
 {
  printf("%c:",65+j);
  scanf("%d",&alloc[i][j]);
 }
}
printf("Enter no.of maximum resources of each resource type by each process:\n");
for(i=0;i<n;i++)
{
 printf("P%d:\n",i);
 for(j=0;j<m;j++)
 {
  printf("%c:",65+j);
  scanf("%d",&max[i][j]);
 }
}
}
void calc_avail()
{
 int i,j,s;
 for(j=0;j<m;j++)
```

```
 {
  s=0;
  for(i=0;i<n;i++)
   s+=alloc[i][j];
  avail[j] = total[j] - s;
 }
}
void calc_need()
{
 int i,j;
 for(i=0;i<n;i++)
  for(j=0;j<m;j++)
   need[i][j]=max[i][j]-alloc[i][j];
}
void print()
{
 int i,j;
 printf("\tAllocation\tMax\tNeed\n\t");
 for(i=0;i<3;i++)
 {
  for(j=0;j<m;j++)
   printf("%3c",65+j);
  printf("\t");
 }
 printf("\n");
 for(i=0;i<n;i++)
 {
  printf("P%d\t",i);
```

```
 for(j=0;j<m;j++)
  printf("%3d",alloc[i][j]);
 printf("\t");
 for(j=0;j<m;j++)
  printf("%3d",max[i][j]);
 printf("\t");
 for(j=0;j<m;j++)
  printf("%3d",need[i][j]);
 printf("\n");
 }
 printf("Available\n");
 for(j=0;j<m;j++)
  printf("%3c",65+j);
 printf("\n");
 for(j=0;j<m;j++)
  printf("%3d",avail[j]);
 printf("\n");
}
int check(int s)
{
 int i,j;
 i = s;
 do
 {
  if(!finish[i])
  {
   for(j=0;j<m;j++)
   {
```

```
    if(need[i][j]>work[j])
     break;
   }
   if(j==m) return i;
  }
  i=(i+1)%n;
 }while(i!=s);
 return -1;
}
void banker()
{
 int i,j,k=0;

 for(i=0;i<n;i++)
  finish[i]=0;
 for(j=0;j<m;j++)
  work[j] = avail[j];
 i=0;
 while((i=check(i))!=-1)
 {
  printf("Process P%d resource granted.\n",i);
  finish[i] = 1;
  for(j=0;j<m;j++)
   work[j] += alloc[i][j];
  printf("finish(");
  for(j=0;j<n;j++)
   printf("%d,",finish[j]);
  printf("\b)\nwork(");
```

```
 for(j=0;j<m;j++)
  printf("%d,",work[j]);
 printf("\b)\n");
 seq[k++]=i;
 i=(i+1)%n;
}
if(k==n)
{
 printf("System is in safe state.\n");
 printf("Safe sequence:");
 for(j=0;j<n;j++)
  printf("P%d ",seq[j]);
}
else
{
 printf("System is not in safe state.");
}
printf("\n");
}
int main()
{
 int i,j,pno;
 accept();
 calc_avail();
 calc_need();
 print();
 banker();
 printf("Enter process no:");
```

```
scanf("%d",&pno);
printf("Enter resource request of process P%d\n",pno);
for(j=0;j<m;j++)
{
 printf("%c:",65+j);
 scanf("%d",&request[j]);
}
for(j=0;j<m;j++)
{
 if(request[j]>need[pno][j])
  break;
}
if(j==m)
{
 for(j=0;j<m;j++)
 {
  if(request[j]>avail[j])
   break;
 }
 if(j==m)
 {
  for(j=0;j<m;j++)
  {
   avail[j]-=request[j];
   alloc[pno][j]+=request[j];
   need[pno][j]-=request[j];
   print();
   banker();
```

```
  }
 }
 else
  printf("Process P%d must wait.\n",pno);
}
else
 printf("Process P%d has exceeded its maximum claim\n",pno);
return 0;
}
```

ASSIGNMENT 2

1) Write a program to simulate Sequential (Contiguous) file allocation method. Assume disk with n number of blocks. Give value of n as input. Write menu driver program with menu options as mentioned above and implement each option.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX 200

typedef struct dir
{
 char fname[20];
 int start;
 struct dir *next;
}NODE;

NODE *first,*last;

int n,fb,bit[MAX];

void init()
{
 int i;

 printf("Enter total no.of disk blocks:");
 scanf("%d",&n);

 fb = n;

 for(i=0;i<10;i++)
```

```
 {
  int k = rand()%n;
  if(bit[k]!=-2)
  {
   bit[k]=-2;
   fb--;
  }
 }
}

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

void show_dir()
{
 NODE *p;
 int i;

 printf("File\tChain\n");

 p = first;
 while(p!=NULL)
 {
  printf("%s\t",p->fname);
  i = p->start;
  while(i!=-1)
  {
   printf("%d->",i);
   i=bit[i];
  }
  printf("NULL\n");

  p=p->next;
 }
}

void create()
{
 NODE *p;
 char fname[20];
 int i,j,nob;

 printf("Enter file name:");
 scanf("%s",fname);
```

```
 printf("Enter no.of blocks:");
 scanf("%d",&nob);

 if(nob>fb)
 {
  printf("Failed to create file %s\n",fname);
  return;
 }

 for(i=0;i<n;i++)
 {
  if(bit[i]==0) break;
 }

 p = (NODE*)malloc(sizeof(NODE));
 strcpy(p->fname,fname);
 p->start=i;
 p->next=NULL;

 if(first==NULL)
  first=p;
 else
  last->next=p;

 last=p;

 fb-=nob;

 j=i+1;
 nob--;

 while(nob>0)
 {
  if(bit[j]==0)
  {
   bit[i]=j;
   i=j;
   nob--;
  }
  j++;
 }

 bit[i]=-1;
 printf("File %s created successully.\n",fname);
}

void delete()
{
```

```
char fname[20];
NODE *p,*q;
int nob=0,i,j;

printf("Enter file name to be deleted:");
scanf("%s",fname);

p = q = first;
while(p!=NULL)
{
 if(strcmp(p->fname,fname)==0)
  break;

 q=p;
 p=p->next;
}

if(p==NULL)
{
 printf("File %s not found.\n",fname);
 return;
}

i = p->start;
while(i!=-1)
{
 nob++;
 j = i;
 i = bit[i];
 bit[j] = 0;
}

fb+=nob;

if(p==first)
 first=first->next;
else if(p==last)
{
 last=q;
 last->next=NULL;
}
else
 q->next = p->next;

free(p);

printf("File %s deleted successfully.\n",fname);
}
```

```
int main()
{
 int ch;

 init();

 while(1)
 {
  printf("1.Show bit vector\n");
  printf("2.Create new file\n");
  printf("3.Show directory\n");
  printf("4.Delete file\n");
  printf("5.Exit\n");
  printf("Enter your choice (1-5):");
  scanf("%d",&ch);

  switch(ch)
  {
  case 1:
   show_bitvector();
   break;
  case 2:
   create();
   break;
  case 3:
   show_dir();
   break;
  case 4:
   delete();
   break;
  case 5:
   exit(0);
  }
 }

 return 0;
}
```

2) Write a program to simulate Linked file allocation method. Assume disk with n number of blocks. Give value of n as input. Write menu driver program with menu options as mentioned above and implement each option.

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MAX 10

typedef struct dir

{

 char fname[20];

int start;

 struct dir *next;

}NODE;

NODE *head,*last;

int n,bit[MAX],fb=0;

void init()

{

 int i;

 printf("Enter total no.of disk blocks:");

 scanf("%d",&n);

 for(i=0;i<n;i++)

 {

bit[i]=rand()%2;

}

}

void show_bitvector()

{
```

```
 int i;
for(i=0;i<n;i++)
 printf("%d ",bit[i]);
 printf("\n");
}
void show_dir()
{
 NODE *temp;
 int i;
for(temp=head;temp!=NULL;temp=temp->next)
{
printf("%d->",temp->start);
}
printf("NULL\n");
}
void create()
{
NODE *p;
 char fname[20];
int i,j,nob;
int fb=0;
printf("Enter file name:");
 scanf("%s",fname);
 printf("Enter no.of blocks:");
 scanf("%d",&nob);
 for(i=0;i<n;i++)
 {
```

```
 if(bit[i]==0)
  fb++;
 }
if(nob>fb)
 {
  printf("Failed to create file %s\n",fname);
return;
 }
 Else
{
for(i=0;i<n;i++)
{
if(bit[i]==0 && nob!=0)
{
p = (NODE*)malloc(sizeof(NODE));
 strcpy(p->fname,fname);
 nob--;bit[i]=1;
 p->start=i;
p->next=NULL;
if(head==NULL)
 head=p;
else
 last->next=p;
last=p;
}
}
printf("File %s created successully.\n",fname);
```

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

3) Write a program to simulate Indexed file allocation method. Assume disk with n number of blocks. Give value of n as input. Write menu driver program with menu options as mentioned above and implement each option

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MAX 200


typedef struct dir

{

        char fname[20];

        int start,length;

        struct dir *next;

}NODE;


NODE *first,*last;

int bit[MAX],n;


void init()

{

        int i;

        printf("Enter total no.of disk blocks:");

        scanf("%d",&n);

        for(i=0;i<n;i++)

        {

                bit[i] =rand()%2;

        }
```

```
}

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

void show_dir()
{
        NODE *p;

        printf("File\tStart\tLength\n");

        p = first;
        while(p!=NULL)
        {
                printf("%s\t%d\t%d\n",
                        p->fname,p->start,p->length);
                p = p->next;
        }
}
```

```
void create()
{
        NODE *p;
        char fname[20];
        int nob,i=0,j=0,start;

        printf("Enter file name:");
        scanf("%s",fname);

        printf("Enter no.of blocks:");
        scanf("%d",&nob);

        while(1)
        {
                while(i<n)
                {
                        if(bit[i]==0) break;
                        i++;
                }

                if(i<n)
                {
                        start = i;
                        j=1;
                        while(j<nob && i<n && bit[i]==0)
                        {
                                i++;j++;
```

```
        }

        if(j==nob)
        {
                p = (NODE*)malloc(sizeof(NODE));
                strcpy(p->fname,fname);
                p->start = start;
                p->length = nob;
                p->next = NULL;

                if(first==NULL)
                        first=p;
                else
                        last->next=p;
                last=p;

                for(j=0;j<nob;j++)
                        bit[j+start]=1;

                printf("File %s created successfully.\n",fname);
                return;
        }
}
else
{
        printf("Fail to create file %s\n",fname);
        return;
```

```
                }
        }
}

void delete()
{
        NODE *p,*q;
        char fname[20];
        int i;

        printf("Enter file to be deleted:");
        scanf("%s",fname);

        p = q = first;
        while(p!=NULL)
        {
                if(strcmp(p->fname,fname)==0)
                        break;

                q = p;
                p = p->next;
        }

        if(p==NULL)
        {
                printf("File %s not found.\n",fname);
                return;
```

```
        }

        for(i=0;i<p->length;i++)
                bit[p->start+i]=0;

        if(p==first)
                first = first->next;
        else if(p==last)
        {
                last=q;
                last->next=NULL;
        }
        else
        {
                q->next = p->next;
        }
        free(p);
        printf("File %s deleted successfully.\n",fname);
}

void main()
{
        int ch;

        init();

        while(1)
```

```
{
        printf("1.Show bit vector\n");
        printf("2.Create new file\n");
        printf("3.Show directory\n");
        printf("4.Delete file\n");
        printf("5.Exit\n");
        printf("Enter your choice (1-5):");
        scanf("%d",&ch);

        switch(ch)
        {
        case 1:
                show_bitvector();
                break;
        case 2:
                create();
                break;
        case 3:
                show_dir();
                break;
        case 4:
                delete();
                break;
        case 5:
                exit(0);
        }
}
```

}

ASSIGNMENT 3

Slot 1

i. Write an OS program to implement FCFS Disk Scheduling algorithm.

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

   // logic for FCFS disk scheduling

   for(i=0;i<n;i++)
   {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];
```

```
    }


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

    return 0;


}
```

ii. Write an OS program to implement SSTF algorithm Disk Scheduling algorithm.

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

    // logic for sstf disk scheduling

       /* loop will execute until all process is completed*/
    while(count!=n)
    {
       int min=1000,d,index;
       for(i=0;i<n;i++)
       {
         d=abs(RQ[i]-initial);
         if(min>d)
         {
            min=d;
            index=i;
         }

       }
       TotalHeadMoment=TotalHeadMoment+min;
       initial=RQ[index];
       // 1000 is for max
```

```
        // you can use any number
        RQ[index]=1000;
        count++;
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

```
//
Enter the number of Request
8
Enter Request Sequence
95 180 34 119 11 123 62 64
Enter initial head Position
50
Total head movement is 236
```

Slot 2

i. Write an OS program to implement SCAN Disk Scheduling algorithm.

```
#include<conio.h>

#include<stdio.h>

int main()

{

int i,j,sum=0,n;

int d[20];

int disk; //loc of head

int temp,max;

int dloc; //loc of disk in array

clrscr();

printf("enter number of location\t");

scanf("%d",&n);
```

```
printf("enter position of head\t");
scanf("%d",&disk);
printf("enter elements of disk queue\n");
for(i=0;i<n;i++)
{
scanf("%d",&d[i]);
}
d[n]=disk;
n=n+1;
for(i=0;i<n;i++) // sorting disk locations
{
for(j=i;j<n;j++)
{
if(d[i]>d[j])
{
temp=d[i];
d[i]=d[j];
d[j]=temp;
}
}


}
max=d[n];
for(i=0;i<n;i++) // to find loc of disc in array
```

```
{
if(disk==d[i]) { dloc=i; break; }
}
for(i=dloc;i>=0;i--)
{
printf("%d -->",d[i]);
}
printf("0 -->");
for(i=dloc+1;i<n;i++)
{
printf("%d-->",d[i]);
}
sum=disk+max;
printf("\nmovement of total cylinders %d",sum);
getch();
return 0;
}
```

iv. Write an OS program to implement C-SCAN algorithm Disk Scheduling algorithm.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
```

```
scanf("%d",&initial);
printf("Enter total disk size\n");
scanf("%d",&size);
printf("Enter the head movement direction for high 1 and for low 0\n");
scanf("%d",&move);

// logic for C-Scan disk scheduling

   /*logic for sort the request array */
for(i=0;i<n;i++)
{
   for( j=0;j<n-i-1;j++)
   {
      if(RQ[j]>RQ[j+1])
      {
         int temp;
         temp=RQ[j];
         RQ[j]=RQ[j+1];
         RQ[j+1]=temp;
      }

   }
}

int index;
for(i=0;i<n;i++)
{
   if(initial<RQ[i])
   {
      index=i;
      break;
   }
}

// if movement is towards high value
if(move==1)
{
   for(i=index;i<n;i++)
   {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];
   }
   //  last movement for max size
   TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
   /*movement max to min disk */
   TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
   initial=0;
   for( i=0;i<index;i++)
   {
```

```
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];

        }
    }
    // if movement is towards low value
    else
    {
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }
        //  last movement for min size
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
        /*movement min to max disk */
        TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
        initial =size-1;
        for(i=n-1;i>=index;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];

        }
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

//
Enter the number of Request
8
Enter the Requests Sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 382

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

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

// size of array
#define n 10

int a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

// Temporary array for slave process
int a2[1000];

int main(int argc, char* argv[])
{

   int pid, np,
      elements_per_process,
      n_elements_recieved;
   // np -> no. of processes
   // pid -> process id

   MPI_Status status;

   // Creation of parallel processes
```

```
MPI_Init(&argc, &argv);

// find out process ID,
// and how many processes were started
MPI_Comm_rank(MPI_COMM_WORLD, &pid);
MPI_Comm_size(MPI_COMM_WORLD, &np);

// master process
if (pid == 0) {
   int index, i;
   elements_per_process = n / np;

   // check if more than 1 processes are run
   if (np > 1) {
      // distributes the portion of array
      // to child processes to calculate
      // their partial sums
      for (i = 1; i < np - 1; i++) {
         index = i * elements_per_process;

         MPI_Send(&elements_per_process,
               1, MPI_INT, i, 0,
               MPI_COMM_WORLD);
         MPI_Send(&a[index],
               elements_per_process,
               MPI_INT, i, 0,
               MPI_COMM_WORLD);
      }
```

```
    // last process adds remaining elements
    index = i * elements_per_process;
    int elements_left = n - index;

    MPI_Send(&elements_left,
             1, MPI_INT,
             i, 0,
             MPI_COMM_WORLD);
    MPI_Send(&a[index],
             elements_left,
             MPI_INT, i, 0,
             MPI_COMM_WORLD);
}

// master process add its own sub array
int sum = 0;
for (i = 0; i < elements_per_process; i++)
    sum += a[i];

// collects partial sums from other processes
int tmp;
for (i = 1; i < np; i++) {
    MPI_Recv(&tmp, 1, MPI_INT,
             MPI_ANY_SOURCE, 0,
             MPI_COMM_WORLD,
             &status);
    int sender = status.MPI_SOURCE;
```

```
            sum += tmp;
        }

        // prints the final sum of array
        printf("Sum of array is : %d\n", sum);
    }
    // slave processes
    else {
        MPI_Recv(&n_elements_recieved,
                 1, MPI_INT, 0, 0,
                 MPI_COMM_WORLD,
                 &status);

        // stores the received array segment
        // in local array a2
        MPI_Recv(&a2, n_elements_recieved,
                 MPI_INT, 0, 0,
                 MPI_COMM_WORLD,
                 &status);

        // calculates its partial sum
        int partial_sum = 0;
        for (int i = 0; i < n_elements_recieved; i++)
            partial_sum += a2[i];

        // sends the partial sum to the root process
        MPI_Send(&partial_sum, 1, MPI_INT,
```

```
            0, 0, MPI_COMM_WORLD);
    }

    // cleans up all MPI state before exit of process
    MPI_Finalize();

    return 0;
}

//
```

Compile and run the program using following code:

```
mpicc program_name.c -o object_file
mpirun -np [number of processes] ./object_file
```

Output:

```
Sum of array is 55
```