PROGRAM:

```
#shell program to calculate the area and circumference of the circle
echo enter the radius
read r
area=`expr 22 / 7 \* $r \* $r`
circumference=`expr 2 \* 22 / 7 \* $r`
echo area= $area
echo circumference= $circumference
```

OUTPUT :

```
enter the radius
5
area= 75
circumference= 30
```

PROGRAM:

```
#shell program to swap two numbers using a temporary variable
echo "enter the two numbers for swapping"
read a b
echo Before swapping
echo A=$a and B=$b c=$a
a=$b b=$c
echo After swapping echo A=$a and B=$b
```

OUTPUT:

```
enter the two numbers for swapping 40 67
Before swapping A= 40 and  B= 67
After swapping A= 67 and B= 40
```

PROGRAM:

```
#shell program to find the gross salary
echo Enter the employee name
read name
echo enter the basic salary
read s
da=`expr $s \* 47 / 100` hra=`expr $s \* 12 / 100`
cca=`expr $s \* 3 / 100`
gross=`expr $s + $hra + $cca + $da`
echo The gross salary of $name is $gross
```

OUTPUT :

```
Enter the employee name Saravanan
Enter The Basic salary 25000
The gross salary of Saravanan is 40500
```

PROGRAM:

/* UNIX SYSTEM CALLS*/

```
#include<stdio.h>
#include<unistd.h>
main()
{
int pid,pid1,pid2;
pid=fork();
if(pid==-1)
{
printf("ERROR IN PROCESS CREATION \n");
exit(1);
}
if(pid!=0)
{
pid1=getpid();
printf("\n The parent process ID is %d\n", pid1);
}
else
{
pid2=getpid();
printf("\n The child process ID is %d\n", pid2);
}
}
```

OUTPUT:

```
The parent process ID is 1315

 The child process ID is 131
```

PROGRAM:
/* IMPLEMENTATION OF IPC USING MESSAGE QUEUE*/

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

struct msg_buffer
{
   long msg_type;
   int data;
};

int isPrime(int n)
{
   if (n <= 1) return 0;
   for (int i = 2; i * i <= n; i++)
      if (n % i == 0) return 0;
   return 1;
}

void sender()
{
   key_t key = ftok("progfile", 65);
   int msgid = msgget(key, 0666 | IPC_CREAT);

   int inputData;
   printf("Enter an integer to send: ");
   scanf("%d", &inputData);

   struct msg_buffer message = {1, inputData};
   msgsnd(msgid, &message, sizeof(message), 0);

   msgrcv(msgid, &message, sizeof(message), 2, 0);
   printf("Received status from receiver: %s\n", message.data ? "Prime" : "Not
Prime");

   msgctl(msgid, IPC_RMID, NULL);
}

void receiver()
{
   key_t key = ftok("progfile", 65);
   int msgid = msgget(key, 0666 | IPC_CREAT);

   struct msg_buffer message;
   msgrcv(msgid, &message, sizeof(message), 1, 0);

   int isPrimeResult = isPrime(message.data);

   message.msg_type = 2;
   message.data = isPrimeResult;
   msgsnd(msgid, &message, sizeof(message), 0);
}

int main()
{
   pid_t pid = fork();

   if (pid == -1)
   {
      fprintf(stderr, "Fork failed\n");
      exit(EXIT_FAILURE);
   }

   pid > 0 ? sender() : receiver();

   return 0;
}
```

OUTPUT:

```
Enter an integer to send:
5
Received status from receiver: Prime
```

PROGRAM:

```
/* FIRST COME FIRST SERVE SCHEDULING*/
#include<stdio.h>

void findWaitingTime(int processes[], int n, int bt[], int wt[])
{
   wt[0] = 0; // Waiting time for the first process is always 0

   // Calculate waiting time for remaining processes
   for (int i = 1; i < n; i++)
   {
      wt[i] = wt[i-1] + bt[i-1];
   }
}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[])
{

   // Calculate turnaround time by adding burst time and waiting time
   for (int i = 0; i < n; i++)
   {
      tat[i] = bt[i] + wt[i];
   }
}

void findGanttChart(int processes[], int n, int bt[], int wt[])
  {
   printf("\nGantt Chart:\n");
   printf("------------------------------------------------------------\n");
   printf("| Process |");
   for (int i = 0; i < n; i++)
   {
      printf("  P%d  |", processes[i]);
   }
   printf("\n------------------------------------------------------------\n");
   printf("|  Time   |");
   for (int i = 0; i < n; i++)
   {
      printf("   %d  |", wt[i]);
   }
   printf("\n------------------------------------------------------------\n");
}

void findAvgTime(int processes[], int n, int bt[])
```

```
  {
    int wt[n], tat[n];
    float total_wt = 0, total_tat = 0;

    // Calculate waiting time of each process
    findWaitingTime(processes, n, bt, wt);

    // Calculate turnaround time of each process
    findTurnAroundTime(processes, n, bt, wt, tat);

    // Display processes along with their respective waiting and turnaround times
    printf("Process   Burst Time   Waiting Time   Turnaround Time\n");
    for (int i = 0; i < n; i++)
    {
        total_wt += wt[i];
        total_tat += tat[i];
        printf("  %d\t\t%d\t\t%d\t\t%d\n", processes[i], bt[i], wt[i], tat[i]);
    }

    // Calculate average waiting and turnaround times
    float avg_wt = total_wt / n;
    float avg_tat = total_tat / n;
    printf("\nAverage Waiting Time: %.2f\n", avg_wt);
    printf("Average Turnaround Time: %.2f\n", avg_tat);

    // Display Gantt Chart
    findGanttChart(processes, n, bt, wt);
}

int main()
{
    int processes[] = {1, 2, 3, 4}; // Process IDs
    int n = sizeof(processes) / sizeof(processes[0]); // Number of processes
    int burst_time[] = {6, 8, 7, 3}; // Burst time of each process

    findAvgTime(processes, n, burst_time);
    return 0;
}
```

SAMPLE OUTPUT :

```
Process   Burst Time   Waiting Time   Turnaround Time
  1             6              0              6
  2             8              6              14
  3             7              14             21
  4                  3                     21                    24

Average Waiting Time: 10.25
Average Turnaround Time: 16.25

Gantt Chart:
-----------------------------------------------------------
| Process |  P1  |  P2  |  P3  |  P4  |
-----------------------------------------------------------
|  Time   |   0 |   6  |  14  |  21  |  24  |
```

PROGRAM:

```
/* SHORTEST JOB FIRST SCHEDULING*/
#include<stdio.h>
#include<conio.h>
main()
{
        int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
        float wtavg, tatavg;
        clrscr();
        printf("\nEnter the number of processes -- ");
        scanf("%d", &n);
        for(i=0;i<n;i++)
        {
                p[i]=i;
                printf("Enter Burst Time for Process %d -- ", i);
                scanf("%d", &bt[i]);
        }
        for(i=0;i<n;i++)
                for(k=i+1;k<n;k++)
                        if(bt[i]>bt[k])
                        {
                                temp=bt[i];
                                bt[i]=bt[k];
                                bt[k]=temp;
                        }
        wt[0] = wtavg = 0;
        temp=p[i]; p[i]=p[k]; p[k]=temp;
        tat[0] = tatavg = bt[0];
        for(i=1;i<n;i++)
        {
                wt[i] = wt[i-1] +bt[i-1];
                tat[i] = tat[i-1] +bt[i]; wtavg = wtavg + wt[i]; tatavg = tatavg +
                tat[i];
        }
        printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t
        TURNAROUND TIME\n");
        for(i=0;i<n;i++)
                printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
         printf("\nAverage Waiting Time -- %f", wtavg/n);
         printf("\nAverage Turnaround Time -- %f", tatavg/n);
         getch();
}
```

SAMPLE OUTPUT:

```
Enter the number of processes --        4
Enter Burst Time for Process 0 --       6
Enter Burst Time for Process 1 --       8
Enter Burst Time for Process 2 --       7
Enter Burst Time for Process 3 --       3

PROCESS       BURST TIME         WAITING TIME       TURNAROUND
TIME
P3                 3                   0                  3
P0                 6                   3                  9
P2                 7                   9                  16
P1                 8                   16                 24

Average Waiting Time --                 7.000000
Average Turnaround Time --             13.000000
```

PROGRAM:

/*ROUND ROBIN SCHEDULING*/

```
#include<stdio.h>
main()
{
        int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;
        float awt=0,att=0,temp=0;
        clrscr();
        printf("Enter the no of processes -- ");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nEnter Burst Time for process %d -- ", i+1);
                scanf("%d",&bu[i]);
                ct[i]=bu[i];
        }
        printf("\nEnter the size of time slice -- ");
        scanf("%d",&t);
        max=bu[0];
        for(i=1;i<n;i++)
                if(max<bu[i])
                        max=bu[i];
        for(j=0;j<(max/t)+1;j++)
                for(i=0;i<n;i++)
                        if(bu[i]!=0)
                                if(bu[i]<=t)
                                {
                                        tat[i]=temp+bu[i];
                                        temp=temp+bu[i];
                                        bu[i]=0;
                                }
                                else
                                {
                                        bu[i]=bu[i]-t;
                                        temp=temp+t;
                                }
        for(i=0;i<n;i++)
        {
                wa[i]=tat[i]-ct[i];
                att+=tat[i];
                awt+=wa[i];
        }
        printf("\nThe Average Turnaround time is -- %f",att/n);
        printf("\nThe Average Waiting time is -- %f ",awt/n);
        printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND
        TIME\n");
        for(i=0;i<n;i++)
                printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);

        getch();

}
```

SAMPLE OUPUT:

```
Enter the no of processes – 3
Enter Burst Time for process 1 --   24
Enter Burst Time for process 2 --    3
Enter Burst Time for process 3 --    3

Enter the size of time slice – 3

The Average Turnaround time is -- 15.666667
The Average Waiting time is --       5.666667

PROCESS      BURST TIME          WAITING TIME       TURNAROUND TIME
1                  24                     6                       30
2                  3                      4                       7
3                  3                      7                       10
```

PROGRAM:

```
/* PRIORITY SCHEDULING*/
#include<stdio.h>
main()
{
    int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp;
    float wtavg, tatavg;
    clrscr();
    printf("Enter the number of processes --- ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
     p[i] = i;
     printf("Enter the Burst Time & Priority of Process %d --- ",i);
     scanf("%d %d",&bt[i], &pri[i]);
    }
    for(i=0;i<n;i++)
      for(k=i+1;k<n;k++)
           if(pri[i] > pri[k])
           {
               temp=p[i];
               p[i]=p[k];
               p[k]=temp;

               temp=bt[i];
               bt[i]=bt[k];
               bt[k]=temp;

               temp=pri[i];
               pri[i]=pri[k];
               pri[k]=temp;

           }
    wtavg = wt[0] = 0;
    tatavg = tat[0] = bt[0];
    for(i=1;i<n;i++)
    {
        wt[i] = wt[i-1] + bt[i-1];
        tat[i] = tat[i-1] + bt[i];

        wtavg = wtavg + wt[i];
        tatavg = tatavg + tat[i];
    }
```

```
        printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING
TIME\tTURNAROUND
                TIME");
        for(i=0;i<n;i++)
                printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d
            ",p[i],pri[i],bt[i],wt[i],tat[i]);

        printf("\nAverage Waiting Time is --- %f",wtavg/n);
        printf("\nAverage Turnaround Time is --- %f",tatavg/n);
        getch();
}
```

SAMPLE OUTPUT:

```
Enter the number of processes --    5
Enter the Burst Time & Priority of Process 0 ---        10      3
Enter the Burst Time & Priority of Process 1 ---        1       1
Enter the Burst Time & Priority of Process 2 ---        2       4
Enter the Burst Time & Priority of Process 3 ---        1       5
Enter the Burst Time & Priority of Process 4 ---        5
PROCESS         PRIORITY    BURST TIME    WAITING TIME   TURNAROUNDTIME
1                   1           1           0                   1
4                   2           5           1                   6
0                   3           10          6                   16
2                   4           2           16                  18
3                   5           1           18                  19

Average Waiting Time is --- 8.200000
Average Turnaround Time is --- 12.000000
```

PROGRAM:

/* PROCESS SYNCHRONIZATION USING SEMAPHORES */

```
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t mutex;
int bal=500;
void* threada(void* arg)
{
   //wait
  sem_wait(&mutex);
  printf("\n  Thread1  Entered\n");
  bal =bal-100;
  printf("Thread1 - A:bal:%d",bal);

//critical section
 sleep(10);
//signal
 printf("\n Thread1 Exit \n");
 sem_post(&mutex);
}
void* threadb(void* arg)
{  //wait
   sem_wait(&mutex);
   printf("\n Thread2  Entered \n");
   bal=bal-50;
   printf("Thread2  bal:%d",bal);
   //critical section
   sleep(10);
   //signal
   printf("\n Thread 2 Exit\n");
   sem_post(&mutex);
}
int main()
{  sem_init(&mutex, 0, 1);
   pthread_t t1,t2;
   pthread_create(&t1,NULL,threada,NULL);
   sleep(2);
   pthread_create(&t2,NULL,threadb,NULL);
   pthread_join(t1,NULL);
   pthread_join(t2,NULL);
   sem_destroy(&mutex);
   return 0;
}
```

OUTPUT:

```
Thread1 Entered
Thread1 - A:bal:400
Thread1 Exit
Thread2 Entered
Thread2 bal:350
Thread 2 Exit
```

PROGRAM:

```
/*BANKER’S ALGORITHM FOR DEADLOCK AVOIDANCE*/
#include <stdio.h>
int main()
{
  // P0, P1, P2, P3, P4 are the Process names here

  int n, m, i, j, k;
  n = 5;                    // Number of processes
  m = 3;                    // Number of resources
  int alloc[5][3] = {{0, 1, 0},  // P0 // Allocation Matrix
                     {2, 0, 0},  // P1
                     {3, 0, 2},  // P2
                     {2, 1, 1},  // P3
                     {0, 0, 2}}; // P4

  int max[5][3] = {{7, 5, 3},  // P0 // MAX Matrix
                   {3, 2, 2},  // P1
                   {9, 0, 2},  // P2
                   {2, 2, 2},  // P3
                   {4, 3, 3}}; // P4

  int avail[3] = {3, 3, 2}; // Available Resources

  int f[n], ans[n], ind = 0;
  for (k = 0; k < n; k++)
  {
    f[k] = 0;
  }
  int need[n][m];
  for (i = 0; i < n; i++)
  {
    for (j = 0; j < m; j++)
      need[i][j] = max[i][j] - alloc[i][j];
  }
  int y = 0;
  for (k = 0; k < 5; k++)
  {
    for (i = 0; i < n; i++)
    {
      if (f[i] == 0)
```

```
         {
            int flag = 0;
            for (j = 0; j < m; j++)
            {
               if (need[i][j] > avail[j])
               {
                  flag = 1;
                  break;
               }
            }
            if (flag == 0)
            {
               ans[ind++] = i;
               for (y = 0; y < m; y++)
                  avail[y] += alloc[i][y];
               f[i] = 1;
            }
         }
      }
   }
   int flag = 1;
   for (int i = 0; i < n; i++)
   {
      if (f[i] == 0)
      {
         flag = 0;
         printf("The following system is not safe");
         break;
      }
   }
   if (flag == 1)
   {
      printf("Following is the SAFE Sequence\n");
      for (i = 0; i < n - 1; i++)
         printf(" P%d ->", ans[i]);
      printf(" P%d", ans[n - 1]);
   }
   return (0);
```

OUTPUT

```
Following is the SAFE Sequence P1 -> P3 -> P4 -> P0 -> P2
```

PROGRAM:

```
/* FIRST FIT, BEST FIT, WORST FIT*/

#include <stdio.h>
#define MAX_PROCESS 10
#define MAX_MEMORY_BLOCK 10

// Function prototypes
void firstFit(int process[], int m, int block[], int n);
void bestFit(int process[], int m, int block[], int n);
void worstFit(int process[], int m, int block[], int n);

int main() {
   int process[MAX_PROCESS], block[MAX_MEMORY_BLOCK];
   int m, n;

   printf("Enter number of processes: ");
   scanf("%d", &m);
   printf("Enter sizes of processes:\n");
   for (int i = 0; i < m; i++)
   {
      scanf("%d", &process[i]);
   }

   printf("Enter number of memory blocks: ");
   scanf("%d", &n);
   printf("Enter sizes of memory blocks:\n");
   for (int i = 0; i < n; i++)
   {
      scanf("%d", &block[i]);
   }

   printf("\nFirst Fit:\n");
   firstFit(process, m, block, n);

   printf("\nBest Fit:\n");
   bestFit(process, m, block, n);

   printf("\nWorst Fit:\n");
   worstFit(process, m, block, n);

   return 0;
}
```

```
void firstFit(int process[], int m, int block[], int n)
   {
    int allocation[m];

    for (int i = 0; i < m; i++)
    {
        allocation[i] = -1;
    }

    for (int i = 0; i < m; i++)
   {
        for (int j = 0; j < n; j++)
        {
            if (block[j] >= process[i])
            {
                allocation[i] = j;
                block[j] -= process[i];
                break;
            }
        }
    }

    printf("Process No.\tProcess Size\tBlock No.\n");
    for (int i = 0; i < m; i++)
     {
        printf("%d\t\t%d\t\t", i + 1, process[i]);
        if (allocation[i] != -1)
        {
            printf("%d\n", allocation[i] + 1);
        }
        else
        {
            printf("Not Allocated\n");
        }
    }
}

void bestFit(int process[], int m, int block[], int n)
 {
    int allocation[m];

    for (int i = 0; i < m; i++)
    {
        allocation[i] = -1;
    }

    for (int i = 0; i < m; i++)
     {
        int bestIdx = -1;
        for (int j = 0; j < n; j++)
         {
            if (block[j] >= process[i])
            {
                if (bestIdx == -1 || block[j] < block[bestIdx])
                {
                    bestIdx = j;
                }
            }
        }
        if (bestIdx != -1)
        {
            allocation[i] = bestIdx;
            block[bestIdx] -= process[i];
        }
    }

    printf("Process No.\tProcess Size\tBlock No.\n");
    for (int i = 0; i < m; i++)
    {
        printf("%d\t\t%d\t\t", i + 1, process[i]);
        if (allocation[i] != -1)
        {
            printf("%d\n", allocation[i] + 1);
        }
        else
        {
            printf("Not Allocated\n");
        }
    }
}

void worstFit(int process[], int m, int block[], int n)
{
    int allocation[m];

    for (int i = 0; i < m; i++)
    {
        allocation[i] = -1;
    }

    for (int i = 0; i < m; i++)
    {
        int worstIdx = -1;
        for (int j = 0; j < n; j++)
        {
            if (block[j] >= process[i])
           {
                if (worstIdx == -1 || block[j] > block[worstIdx])
                {
                    worstIdx = j;
                }
            }
        }
        if (worstIdx != -1)
        {
            allocation[i] = worstIdx;
            block[worstIdx] -= process[i];
        }
    }

    printf("Process No.\tProcess Size\tBlock No.\n");
    for (int i = 0; i < m; i++)
     {
        printf("%d\t\t%d\t\t", i + 1, process[i]);
        if (allocation[i] != -1)
       {
            printf("%d\n", allocation[i] + 1);
        }
        else
        {
            printf("Not Allocated\n");
        }
    }
}
```

OUTPUT:

```
Enter number of processes: 4
Enter sizes of processes:
100
200
300
```

```
400
Enter number of memory blocks: 5
Enter sizes of memory blocks:
150
350
200
500
100

First Fit:
Process No.    Process Size    Block No.
1                100             1
2                200             2
3                300             4
4                400             Not Allocated

Best Fit:
Process No.    Process Size    Block No.
1                100             1
2                200             3
3                300             2
4                400             4

Worst Fit:
Process No.    Process Size    Block No.
1                100             5
2                200             4
3                300             2
4                400             4
```

PROGRAM:

```
/* PAGE REPLACEMENT ALGORITHM*/
/* FIRST IN FIRST OUT*/
#include<stdio.h>
#include<conio.h>
main()
{
    int i, j, k, f, pf=0, count=0, rs[25], m[10], n;
    clrscr();
    printf("\n Enter the length of reference string -- ");
    scanf("%d",&n);
    printf("\n Enter the reference string -- ");
    for(i=0;i<n;i++)
        scanf("%d",&rs[i]);
    printf("\n Enter no. of frames -- ");
    scanf("%d",&f);
    for(i=0;i<f;i++)
        m[i]=-1;

    printf("\n The Page Replacement Process is -- \n");
    for(i=0;i<n;i++)
    {
        for(k=0;k<f;k++)
        {
             if(m[k]==rs[i])
                 break;
        }
        if(k==f)
        {
            m[count++]=rs[i];
            pf++;
        }

        for(j=0;j<f;j++)
            printf("\t%d",m[j]);
        if(k==f)
            printf("\tPF No. %d",pf);
        printf("\n");
        if(count==f)
            count=0;
    }
    printf("\n The number of Page Faults using FIFO are %d",pf);
    getch();
}
```

OUTPUT:

```
Enter the length of reference string – 20
Enter the reference string --   7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter no. of frames --   3

The Page Replacement Process is –
7       -1      -1      PF No. 1
7       0       -1      PF No. 2
7       0       1       PF No. 3
2       0       1       PF No. 4
2       0       1
2       3       1       PF No. 5
2       3       0       PF No. 6
4       3       0       PF No. 7
4       2       0       PF No. 8
4       2       3       PF No. 9
0       2       3       PF No. 10
0       2       3
0       2       3
0       1       3       PF No. 11
0       1       2       PF No. 12
0       1       2
0       1       2
7       1       2       PF No. 13
7       0       2       PF No. 14
7       0       1       PF No. 15

The number of Page Faults using FIFO are 15
```

PROGRAM:

```
/* LEAST RECENTLY USED*/
#include<stdio.h>
#include<conio.h>
main()
{
        int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1;
        clrscr();
        printf("Enter the length of reference string -- ");
        scanf("%d",&n);
        printf("Enter the reference string -- ");
        for(i=0;i<n;i++)
        {
                scanf("%d",&rs[i]);
                flag[i]=0;
        }
        printf("Enter the number of frames -- ");
        scanf("%d",&f);
        for(i=0;i<f;i++)
        {
                count[i]=0;
                m[i]=-1;
        }
        printf("\nThe Page Replacement process is -- \n");
        for(i=0;i<n;i++)
        {
                for(j=0;j<f;j++)
                {
                        if(m[j]==rs[i])
                        {
                                flag[i]=1;
                                count[j]=next;
                                next++;
                        }

                }
        if(flag[i]==0)
        {
                if(i<f)
                {
                        m[i]=rs[i];
                        count[i]=next;
                        next++;
                }
        else
        {
            min=0;
            for(j=1;j<f;j++)
                if(count[min] > count[j])
                    min=j;
            m[min]=rs[i];
            count[min]=next;
            next++;
    }
    pf++;
}
for(j=0;j<f;j++)
    printf("%d\t", m[j]);
        if(flag[i]==0)
            printf("PF No. -- %d" , pf);
    printf("\n");
    }
    printf("\nThe number of page faults using LRU are %d",pf);
    getch();
}
```

OUTPUT:

```
Enter the length of reference string -- 20
Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter the number of frames -- 3

The Page Replacement process is --
7       -1      -1      PF No. -- 1
7       0       -1      PF No. -- 2
7       0       1       PF No. -- 3
2       0       1       PF No. -- 4
2       0       1
2       0       3       PF No. -- 5
2       0       3
4       0       3       PF No. – 6
4       0       2       PF No. – 7
4       3       2       PF No. – 8
0       3       2       PF No. – 9
0       3       2
0       3       2
1       3       2       PF No. – 10
1       3       2
```

1	0	2	PF No. – 11
1	0	2	
1	0	7	PF No. – 12
1	0	7	
1	0	7	

The number of page faults using LRU are 12

PROGRAM:

```
/* OPTIMAL PAGE REPLACMENT ALGORITHM */
#include<stdio.h>
int n;
main()
{
        int seq[30],fr[5],pos[5],find,flag,max,i,j,m,k,t,s;
        int count=1,pf=0,p=0;
        float pfr;
        clrscr();
        printf("Enter maximum limit of the sequence: ");
        scanf("%d",&max); printf("\nEnter the sequence: ");
        for(i=0;i<max;i++)
                scanf("%d",&seq[i]);
                printf("\nEnter no. of frames: ");
                scanf("%d",&n);
                fr[0]=seq[0];
                pf++;
                printf("%d\t",fr[0]);
                i=1;
                while(count<n)
                {
                        flag=1;
                        p++;
                        for(j=0;j<i;j++)
                        {
                                if(seq[i]==seq[j])
                                flag=0;
                        }
                        if(flag!=0)
                        {
                                fr[count]=seq[i];
                                printf("%d\t",fr[count]);
                                count++;
                                pf++;
                        }
                        i++;
                }
                printf("\n");
                for(i=p;i<max;i++)
                {
                        flag=1;
                        for(j=0;j<n;j++)
                        {
```

```
        if(seq[i]==fr[j])
        flag=0;
}
if(flag!=0)
{
        for(j=0;j<n;j++)
        {
                m=fr[j];
                for(k=i;k<max;k++)
                {
                        if(seq[k]==m)
                        {
                                pos[j]=k;
                                break;

                        }
                        else
                                pos[j]=1;

                }
        }
        for(k=0;k<n;k++)
        {
                if(pos[k]==1)
                        flag=0;
        }
        if(flag!=0)
                s=findmax(pos);
        if(flag==0)
        {
                for(k=0;k<n;k++)
                {
                        if(pos[k]==1)
                        {
                                s=k;
                                break;
                        }
                }
        }
        fr[s]=seq[i];
        for(k=0;k<n;k++)
                printf("%d\t",fr[k]);
        pf++;
        printf("\n");
```

```
                }
            }
            pfr=(float)pf/(float)max;
            printf("\nThe no. of page faults are %d",pf);
            printf("\nPage fault rate %f",pfr);
            getch();
}

            int findmax(int a[])
            {
                int max,i,k=0;
                max=a[0];
                for(i=0;i<n;i++)
                {
                    if(max<a[i])
                    {
                        max=a[i];
                        k=i;
                    }
                }
                return k;
            }
```

OUTPUT:

```
Enter number of page references -- 10
Enter the reference string --    1 2 3 4 5 2 5 2 5 1 4 3
Enter the available no. of frames -- 3

The Page Replacement Process is –

1       -1      -1      PF No. 1
1       2       -1      PF No. 2
1       2       3       PF No. 3
4       2       3       PF No. 4
5       2       3       PF No. 5
5       2       3
5       2       3
5       2       1       PF No. 6
5       2       4       PF No. 7
5       2       3       PF No. 8

Total number of page faults --          8
```

PROGRAM:

/* SEQUENTIAL FILE ALLOCATION */

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

struct fileTable
{
        char name[20];
        int sb, nob;
}ft[30];


void main()
{
        int i, j, n; char s[20]; clrscr();
        printf("Enter no of files:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nEnter file name %d          :",i+1);
                scanf("%s",ft[i].name);
                printf("Enter starting block of file %d          :",i+1);
                scanf("%d",&ft[i].sb);
                printf("Enter no of blocks in file %d :",i+1);
                scanf("%d",&ft[i].nob);
        }
        printf("\nEnter the file name to be searched -- ");
        scanf("%s",s);
        for(i=0;i<n;i++)
                if(strcmp(s, ft[i].name)==0)
                        break;
        if(i==n)
                printf("\nFile Not Found");
        else
        {
        printf("\nFILE NAME START BLOCK NO OF BLOCKS BLOCKS
        OCCUPIED\n"); printf("\n%s\t\t%d\t\t%d\t",ft[i].name,ft[i].sb,ft[i].nob);
        for(j=0;j<ft[i].nob;j++)
        printf("%d, ",ft[i].sb+j);
        }
        getch();
}
```

OUTPUT:

```
Enter no of files :3

Enter file name 1 :A
Enter starting block of file 1 :85
Enter no of blocks in file 1 :6

Enter file name 2 :B
Enter starting block of file 2 :102
Enter no of blocks in file 2 :4

Enter file name 3 :C
Enter starting block of file 3 :60
Enter no of blocks in file 3 :4


Enter the file name to be searched -- B

FILE NAME       START BLOCK       NO OF BLOCKS          BLOCKS OCCUPIED
B                   102                    4               102, 103, 104, 105
```

PROGRAM:

```
/** INDEXED FILE ALLOCATION */
#include<stdio.h>
#include<conio.h>

struct fileTable
{
        char name[20];
        int nob, blocks[30];
}ft[30];

void main()
{
        int i, j, n; char s[20]; clrscr();
        printf("Enter no of files      :");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nEnter file name %d:",i+1);
                scanf("%s",ft[i].name);
                printf("Enter no of blocks in file %d :",i+1);
                scanf("%d",&ft[i].nob);
                printf("Enter the blocks of the file :");
                for(j=0;j<ft[i].nob;j++)
                        scanf("%d",&ft[i].blocks[j]);
        }

        printf("\nEnter the file name to be searched -- ");
        scanf("%s",s);
        for(i=0;i<n;i++)
                 if(strcmp(s, ft[i].name)==0)
                        break;
         if(i==n)
                 printf("\nFile Not Found");

        else
        {
                printf("\nFILE NAME NO OF BLOCKS BLOCKS OCCUPIED");
                printf("\n %s\t\t%d\t",ft[i].name,ft[i].nob); for(j=0;j<ft[i].nob;j++)
                printf("%d, ",ft[i].blocks[j]);
        }
        getch();
}
```

OUTPUT:

```
Enter no of files:        2

Enter file 1 : A
Enter no of blocks in file 1:     4
Enter the blocks of the file 1:  12 23 9 4

Enter file 2 : G
Enter no of blocks in file 2:     5
Enter the blocks of the file 2:  88 77 66 55 44

Enter the file to be searched : G


FILE NAME          NO OF BLOCKS                    BLOCKS OCCUPIED
  G                         5                          88, 77, 66, 55, 44
```

PROGRAM:

```
/* LINKED FILE ALLOCATION */
#include<stdio.h>
#include<conio.h>

struct fileTable
{
        char name[20];
        int nob;
        struct block *sb;
 }ft[30];

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


void main()
{
        int i, j, n;
        char s[20];
        struct block *temp;
        clrscr();
        printf("Enter no of files      :");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nEnter file name %d            :",i+1);
                scanf("%s",ft[i].name);
                printf("Enter no of blocks in file %d :",i+1);
                scanf("%d",&ft[i].nob);
                ft[i].sb=(struct block*)malloc(sizeof(struct block));
                temp = ft[i].sb;
                printf("Enter the blocks of the file :");
                scanf("%d",&temp->bno);
                temp->next=NULL;

                for(j=1;j<ft[i].nob;j++)
                {
                        temp->next = (struct block*)malloc(sizeof(struct block));
                        temp = temp->next;
                        scanf("%d",&temp->bno);
                }
                temp->next = NULL;
        }
        printf("\nEnter the file name to be searched -- ");
        scanf("%s",s);
        for(i=0;i<n;i++)
                if(strcmp(s, ft[i].name)==0)
                        break;

        if(i==n)
                printf("\nFile Not Found");
        else
        {
                printf("\nFILE NAME NO OF BLOCKS BLOCKS OCCUPIED");
                printf("\n %s\t\t%d\t",ft[i].name,ft[i].nob);
                temp=ft[i].sb;
                for(j=0;j<ft[i].nob;j++)
                {
                        printf("%d -> ",temp->bno);
                        temp = temp->next;
                }
        }
        getch();
}
```

OUTPUT:

```
Enter no of files:        2

Enter file 1 : A
Enter no of blocks in file 1:      4
Enter the blocks of the file 1: 12 23 9 4

Enter file 2 : G
Enter no of blocks in file 2:      5
Enter the blocks of the file 2:  88 77 66 55 44

Enter the file to be searched : G

FILE NAME        NO OF BLOCKS                    BLOCKS OCCUPIED
G                        5                        88 -> 77-> 66-> 55-> 44
```

PROGRAM:

```
/** SINGLE LEVEL DIRECTORY ORGANIZATION*/
#include<stdio.h>
struct
{
    char dname[10],fname[10][10];
    int fcnt;
}dir;


void main()
{
    int i,ch;
    char f[30];
    clrscr();
    dir.fcnt = 0;
    printf("\nEnter name of directory -- ");
    scanf("%s", dir.dname);
    while(1)
    {
        printf("\n\n1. Create File\t2. Delete File\t3. Search File \n4. Display
    Files\t
              5.Exit\nEnter your choice -- ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                printf("\nEnter the name of the file -- ");
                scanf("%s",dir.fname[dir.fcnt]);
                dir.fcnt++;
                break;
            case 2:
                printf("\nEnter the name of the file -- ");
                scanf("%s",f);
                for(i=0;i<dir.fcnt;i++)
                {
                    if(strcmp(f, dir.fname[i])==0)
                    {
                        printf("File %s is deleted ",f);
                        strcpy(dir.fname[i],dir.fname[dir.fcnt-
                        1]);
                        break;
                    }
                }
```

```
                        if(i==dir.fcnt)
                                printf("File %s not found",f);
                        else
                                dir.fcnt--;
                        break;
                case 3:
                        printf("\nEnter the name of the file -- ");
                        scanf("%s",f);
                        for(i=0;i<dir.fcnt;i++)
                        {
                                if(strcmp(f, dir.fname[i])==0)
                                {
                                        printf("File %s is found ", f);
                                        break;
                                }
                        }
                        if(i==dir.fcnt)
                                printf("File %s not found",f);
                        break;
                case 4:
                        if(dir.fcnt==0)
                                printf("\nDirectory Empty");

                        else
                        {
                                printf("\nThe Files are -- ");
                                for(i=0;i<dir.fcnt;i++)
                                        printf("\t%s",dir.fname[i]);
                        }
                        break;
                default:
                        exit(0);
        }

    }
    getch();
}
```

OUTPUT:

```
Enter name of directory -- CSE
1. Create File       2. Delete File 3. Search File  4. Display Files      5. Exit
Enter your choice – 1
Enter the name of the file -- A

1. Create File       2. Delete File 3. Search File 4. Display Files     5. Exit
Enter your choice – 1

Enter the name of the file -- B

1. Create File       2. Delete File 3. Search File 4. Display Files     5. Exit
Enter your choice – 1

Enter the name of the file -- C
1. Create File       2. Delete File 3. Search File 4. Display Files     5. Exit
Enter your choice – 4

The Files are -- A B C

1. Create File       2. Delete File 3. Search File 4. Display Files     5. Exit
Enter your choice – 3

Enter the name of the file – ABC
File ABC not found

1. Create File       2. Delete File 3. Search File 4. Display Files     5. Exit
Enter your choice – 2

Enter the name of the file – B
File B is deleted

1. Create File       2. Delete File 3. Search File 4. Display Files     5. Exit
Enter your choice – 5
```

PROGRAM:

```
/* HIERARCHICAL DIRECTORY ORGANIZATION*/
#include<stdio.h>
struct
{
        char dname[10],fname[10][10];
        int fcnt;
}dir[10];

void main()
{
        int i,ch,dcnt,k;
        char f[30], d[30];
        clrscr();
        dcnt=0;

        while(1)
        {

                printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
                printf("\n4. Search File\t\t5. Display\t6. Exit\t Enter your choice --
        ");
                scanf("%d",&ch);
                 switch(ch)
                {

                        case 1:
                                printf("\nEnter name of directory -- ");
                                scanf("%s", dir[dcnt].dname);
                                dir[dcnt].fcnt=0;
                                dcnt++;
                                printf("Directory created");
                                break;
                        case 2:
                                printf("\nEnter name of the directory -- ");
                                scanf("%s",d);
                                for(i=0;i<dcnt;i++)
                                        if(strcmp(d,dir[i].dname)==0)
                                        {
                                                 printf("Enter name of the file -- ");
                                                scanf("%s",dir[i].fname[dir[i].fcnt]);
                                                dir[i].fcnt++;
                                                printf("File created");
                                                break;
```

```
                    }
                if(i==dcnt)
                    printf("Directory %s not found",d);
                break;

            case 3:
                printf("\nEnter name of the directory -- ");
                scanf("%s",d);
                for(i=0;i<dcnt;i++)
                {
                    if(strcmp(d,dir[i].dname)==0)
                    {
                        printf("Enter name of the file -- ");
                        scanf("%s",f);
                        for(k=0;k<dir[i].fcnt;k++)
                        {
                            if(strcmp(f, dir[i].fname[k])==0)
                            {
                            printf("File %s is deleted ",f);
                            dir[i].fcnt--;
                            strcpy(dir[i].fname[k],dir[i].fnam
                            e[dir[i].fcnt]);
                            goto jmp;
                            }
                        }
                        printf("File %s not found",f);
                        goto jmp;
                    }
                }
                printf("Directory %s not found",d);
                jmp : break;

            case 4:
                printf("\nEnter name of the directory -- ");
                scanf("%s",d);
                for(i=0;i<dcnt;i++)
                {
                    if(strcmp(d,dir[i].dname)==0)
                    {
                        printf("Enter the name of the file -- ");
                        scanf("%s",f);
                        for(k=0;k<dir[i].fcnt;k++)
                        {
                            if(strcmp(f, dir[i].fname[k])==0)
                            {
                                printf("File %s is found
                            ",f);
                                goto jmp1;
                            }
                        }
                        printf("File %s not found",f);
                        goto jmp1;
                    }
                }
                printf("Directory %s not found",d);
                jmp1: break;
            case 5:
                if(dcnt==0)
                    ("\nNo Directory's ");
                else
                {
                    printf("\nDirectory\tFiles");
                    for(i=0;i<dcnt;i++)
                    {
                        printf("\n%s\t\t",dir[i].dname);
                        for(k=0;k<dir[i].fcnt;k++)
                            printf("\t%s",dir[i].fname[k]);
                    }
                }
                break;
            default:
                exit(0);
        }
    }
    getch();
}
```

OUTPUT:

```
1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display    6. Exit
Enter your choice --    1

Enter name of directory -- DIR1
Directory created

1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display    6. Exit
]
Enter your choice --    1
```

```
          Enter name of directory -- DIR2
          Directory created

          1. Create Directory     2. Create File  3. Delete File 4. Search File   5. Display       6.
Exit
          Enter your choice --    2

          Enter name of the directory – DIR1
          Enter name of the file       --       A1
          File created

          1. Create Directory     2. Create File  3. Delete File 4. Search File   5. Display       6.
Exit
          Enter your choice --    2

          Enter name of the directory – DIR1
          Enter name of the file       --       A2
          File created

          1. Create Directory     2. Create File  3. Delete File  4. Search File  5. Display       6.
Exit
           Enter your choice --   2

          Enter name of the directory – DIR2
          Enter name of the file       --       B1
          File created


          1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6. Exit
          Enter your choice --        5

          Directory           Files
          DIR1                A1  A2
          DIR2                B1


          1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6. Exit
          Enter your choice --        4

          Enter name of the directory – DIR Directory not found

          1. Create Directory     2. Create File  3. Delete File  4. Search File  5. Display       6.
Exit
          Enter your choice --    3
          Enter name of the directory – DIR1
          Enter name of the file -- A2
          File A2 is deleted

          1. Create Directory   2. Create File 3. Delete File 4. Search File   5. Display      6. Exit
          Enter your choice -- 6
```