# DISTRIBUTED OPERATING SYSTEMS LAB MANUAL

| | | |
|---|---|---|
| **Year** | **:** | **2016 - 2017** |
| **Course Code** | **:** | BCS102 |
| **Regulations** | **:** | **IARE - R16** |
| **Semester** | **:** | **II** |
| **Branch** | **:** | **CSE** |

Prepared By

Dr. K. Rajendra Prasad,Professor

Mr. P. Anjaiah,Assistant Professor

**Department of Computer Science & Engineering**
**INSTITUTE OF AERONAUTICAL ENGINEERING**
**Dundigal – 500 043, Hyderabad**

# VISION AND MISSION OF THE DEPARTMENT

## VISION

The Vision of the department is to produce competent graduates suitable for industries and organizations at global level including research and development with Social responsibility.

## MISSION

To provide an open environment to foster professional and personal growth with a strong theoretical and practical background having an emphasis on hardware and software development making the graduates industry ready with social ethics.

Further the Department is to provide training and to partner with   Global entities in education and research.

# INSTITUTE OF AERONAUTICAL ENGINEERING

**(Autonomous)**

**DUNDIGAL – 500 043, HYDERABAD**

**COMPUTER SCIENCE AND ENGINEERING**

| **M.TECH-PROGRAM OUTCOMES(POS)** | |
|---|---|
| PO1 | **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering |
| PO2 | **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences. |
| PO3 | **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental |
| PO4 | **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions. |
| PO5 | **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations. |
| PO6 | **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice. |
| PO7 | **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development. |
| PO8 | **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice. |
| PO9 | **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings. |
| PO10 | **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions. |
| PO11 | **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments. |
| PO12 | **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change. |
| **Program Specific** | |
| PSO1 | **Professional Skills:** The ability to research, understand and implement computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient analysis and design of computer-based systems of varying |

| | |
|---|---|
| PSO2 | **Problem-Solving Skills:** The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success. |
| PSO3 | **Successful Career and Entrepreneurship:** The ability to employ modern computer languages, environments, and platforms in creating innovative career paths, to be an entrepreneur, and a zest for higher studies. |

# OBJECTIVES OF THE DEPARTMENT

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### Program Educational Objectives (PEOs)

A Post Graduate of the Computer Science and Engineering Program should:

| | |
|---|---|
| **PEO – I** | Students will establish themselves as effective professionals by solving real problems through the use of computer science knowledge and with attention to team work, effective communication, critical thinking and problem solving skills. |
| **PEO–II** | Students will develop professional skills that prepare them for immediate employment and for life-long learning in advanced areas of computer science and related fields. |
| **PEO– III** | Students will demonstrate their ability to adapt to a rapidly changing environment by having learned and applied new skills and new technologies. |
| **PEO– IV** | Students will be provided with an educational foundation that prepares them for excellence, leadership roles along diverse career paths with encouragement to professional ethics and active participation needed for a successful career. |

### Program Specific Outcomes (PSO's)

| | |
|---|---|
| **PSO – I** | **Professional Skills:** The ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient design of computer-based systems of varying complexity. |
| **PSO – II** | **Problem-Solving Skills:** The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success. |
| **PSO – III** | **Successful Career and Entrepreneurship:** The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, and a zest for higher studies. |

## Attainment of Program Outcomes and Program Specific Outcomes

| S.No | Experiment | Program Outcomes Attained | Program Specific Outcomes Attained |
|---|---|---|---|
| 1 | Simulate the following CPU scheduling algorithms<br>a) Round Robin b) SJF c) FCFS d) Priority | PO1,PO5 | PSO1 |
| 2 | Simulate all file allocation strategies<br>a) Sequential b) Indexed c) Linked | PO2, PO5 | PSO1 |
| 3 | Implement process strategies: creation of child, zombie, orphan process | PO1,PO6 | PSO1 |
| 4 | Implement file organization strategies<br>a) single level b) Two level c) Hierarchical | PO1,PO6 | PSO1 |
| 5 | Simulate Bankers Algorithm for Dead Lock Avoidance | PO3,PO7 | PSO3 |
| 6 | Simulate Bankers Algorithm for Dead Lock Prevention | PO6,PO8 | PSO1, PSO2 |
| 7 | Simulate all page replacement algorithms<br>a) FIFO b) LRU c) LFU | PO5,PO8 | PSO1, PSO2 |
| 8 | Implement shared memory and semaphore concepts for inter process communication | PO6,PO7 | PSO1,PSO3 |

## Mapping Course Objectives Leading To the Achievement of Program Outcomes

| Course Objectives | Program Outcomes | | | | | | | | | | | | Program Specific Outcomes | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 | PSO1 | PSO2 | PSO3 |
| I | √ | √ | √ | | | | √ | √ | | | | | √ | √ | √ |
| II | √ | √ | √ | √ | √ | | | | | | | | √ | √ | √ |
| III | √ | | √ | | √ | √ | | √ | | | | | √ | √ | √ |

# SYLLABUS

## DISTRIBUTED OPERATING SYSTEM LABORATORY

**II Semester: CSE**

| Course Code | Category | Hours / Week | | | Credits | Maximum Marks | | |
|---|---|---|---|---|---|---|---|---|
| | | L | T | P | C | CIA | SEE | Total |
| BCS102 | Core | - | - | 3 | 2 | 30 | 70 | 100 |
| Contact Classes: Nil | Total Tutorials: Nil | Total Practical Classes: 36 | | | | Total Classes: 36 | | |

**OBJECTIVES:**

**The course should enable the students to:**

I. Understand the design aspects of operating system.
II. Exposure on usage of various operating systems.
III. Design modern distributed system components.

**LIST OF EXPERIMENTS**

| Week | Experiment |
|---|---|
| **Week-1** | **CPU SCHEDULING ALGORITHMS** |
| Simulate the following CPU scheduling algorithms<br>a) Round Robin b) SJF c) FCFS d) Priority | |
| **Week-2** | **FILE ALLOCATION STRATEGIES** |
| Simulate all file allocation strategies<br>a) Sequential b) Indexed c) Linked | |
| **Week-3** | **PROCESS MANAGEMENT** |
| Implement process strategies: creation of child, zombie, orphan process | |
| **Week-4** | **FILE ORGANIZATION STRATEGIES** |
| Implement file organization strategies<br>a) single level b) Two level c) Hierarchical | |
| **Week-5** | **DEAD LOCK AVOIDANCE** |
| Simulate Bankers Algorithm for Dead Lock Avoidance | |
| **Week-6** | **DEAD LOCK PREVENTION** |
| Simulate Bankers Algorithm for Dead Lock Prevention | |
| **Week-7** | **PAGE REPLACEMENT ALGORITHMS** |
| Simulate all page replacement algorithms<br>a) FIFO b) LRU c) LFU | |

| **Week-8** | **SHARED MEMORY AND SEMAPHORE** |
|---|---|
| Implement shared memory and semaphore concepts for inter process communication | |
| **Reference Books:** | |
| Andrew S. Tanenbaum, "Distributed Operating System", PHI, 1st Edition, 1994. | |
| **Web References:** | |
| 1. www.cs.put.poznan.pl/pawelw/sus/dcs07.doc<br>2. https://developer.apple.com/library/mac/documentation | |
| **SOFTWARE AND HARDWARE REQUIREMENTS FOR 18 STUDENTS:**<br>**SOFTWARE:** Turbo C/ J2SE<br>**HARDWARE:** 18 numbers of Intel Desktop Computers with 2 GB RAM | |

# INDEX

# EXPERIMENTS

## 1 A) ROUND ROBIN CPU SCHEDULING ALGORITHM

For round robin scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the size of the time slice. Time slices are assigned to each process in equal portions and in circular order, handling all processes execution. This allows every process to get an equal chance. Calculate the waiting time and turnaround time of each of the processes accordingly

**PROGRAM**

**A) ROUND ROBIN CPU SCHEDULING ALGORITHM**

```
#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();
 }
```

**INPUT**

```
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
```

**OUTPUT**

```
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
Average Waiting Time--   17.000000
Average Turnaround Time -- 27.000000
```

**B) SJF CPU SCHEDULING ALGORITHM**

For SJF scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. Arrange all the Jobs in order with respect to their burst times. There may be two jobs in queue with the same execution time, and then FCFS approach is to be performed. Each process will be executed according to the length of its burst time. Then calculate the waiting time and turnaround time of each of the processes accordingly

**PROGRAM**

```
#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;

                     temp=p[i];
                     p[i]=p[k];
                     p[k]=temp;
               }
             wt[0] = wtavg = 0;
             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();
    }
```

**INPUT**

```
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
```

**OUTPUT**

```
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
```

## C). FCFS CPU SCHEDULING ALGORITHM

For FCFS scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. The scheduling is performed on the basis of arrival time of the processes irrespective of their other parameters.

Each process will be executed according to its arrival time. Calculate the waiting time and turnaround time of each of the processes accordingly.

**PROGRAM**

```
      #include<stdio.h>
      #include<conio.h>
      main()
      {
              int     bt[20],     wt[20],
              tat[20], i, n; float wtavg,
              tatavg;
              clrscr();
              printf("\nEnter the number of processes -- ");
              scanf("%d", &n);
              for(i=0;i<n;i++)
              {
                printf("\nEnter Burst Time for Process %d -- ", i);
                scanf("%d", &bt[i]);
              }
              wt[0] = wtavg = 0;
              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("\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", i, bt[i], wt[i], tat[i]);
              printf("\nAverage Waiting Time -- %f", wtavg/n);
              printf("\nAverage Turnaround Time -- %f", tatavg/n);
              getch();
   }
```

**INPUT**

```
Enter the number of processes --3
Enter Burst Time for Process 0 --         24
Enter Burst Time for Process 1 --         3
Enter Burst Time for Process 2 --         3
```

**OUTPUT**

```
PROCESS              BURST TIME    WAITING TIME       TURNAROUND TIME
   P0           24                 0                  24
   P1           3                  24                 27
   P2           3                  27                 30
Average Waiting Time--   17.000000
Average Turnaround Time --      27.000000
```

## D) PRIORITY CPU SCHEDULING ALGORITHM

For priority scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the priorities. Arrange all the jobs in order with respect to their priorities. There may be two jobs in queue with the same priority, and then FCFS approach is to be performed. Each process will be executed according to its priority. Calculate the waiting time and turnaround time of each of the processes accordingly.

**PROGRAM**

```
#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 \t PRIORITY\tBURSTTIME \tWAITING TIME \t TURNAROUND 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();
  }
```

**INPUT**

```
    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       2
```

**OUTPUT**

```
    PROCESS   PRIORITY  BURST TIME  WAITING TIME     TURNAROUND TIME
    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
```

## 2A) SEQUENTIAL FILE ALLOCATION

In this file organization, the records of the file are stored one after another both physically and logically. That is, record with sequence number 16 is located just after the 15th record. A record of a sequential file can only be accessed by reading all the previous records.

**PROGRAM**

```
#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 NOOFBLOCKS 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();
  }
```

**INPUT:**

```
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
```

**OUTPUT:**

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

**2B) INDEXED FILE ALLOCATION**

Indexed file allocation strategy brings all the pointers together into one location: an index block. Each file has its own index block, which is an array of disk-block addresses. The ith entry in the index block points to the ith block of the file. The directory contains the address of the index block. To find and read the ith block, the pointer in the ith index-block entry is used.

**PROGRAM**

```
#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();
}
```

**INPUT:**

```
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
```

**OUTPUT:**

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

## 2C). LINKED FILE ALLOCATION

With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block.

**PROGRAM**

```
#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();
}
```

**INPUT:**

```
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
```

**OUTPUT:**

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

## 3. Program to implement process strategies: creation of child, zombie and orphan process

```
#include<stdio.h>
main()
{
   int id;

   printf("Before fork()\n");
   id=fork();

   if(id==0)
   {
      printf("Child has started: %d\n ",getpid());
      printf("Parent of this child : %d\n",getppid());
      printf("child prints 1 item :\n ");
      sleep(10);
      printf("child prints 2 item :\n");
   }
   else
   {
      printf("Parent has started: %d\n",getpid());
      printf("Parent of the parent proc : %d\n",getppid());
   }

   printf("After fork()");
}
```

**OUTPUT**
***********

```
[04mca58@LINTEL 04mca58]$  cc orphan.c
[04mca58@LINTEL 04mca58]$  ./a.out
Before fork()
Parent has started: 2899
Child has started: 2900
Parent of this child : 2899
child prints 1 item :
Parent of the parent proc : 616
After fork()
[04mca58@LINTEL 04mca58]$   child prints 2 item :
After fork()
```

**PROGRAM FOR ZOMBIE PROCESS**

```
#include<stdio.h>
main()
{
   int id;
   id=fork();
   if(id>0)
   {
      printf("Parent will sleep");
      sleep(10);
   }
   if(id==0)
      printf("I am child");
}
```

OUTPUT
***********

```
[04mca58@LINTEL 04mca58]$  cc zombie.c
[04mca58@LINTEL 04mca58]$  ./a.out
I am child
Parent will sleep
[04mca58@LINTEL 04mca58]$
```

```
//same program different code
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>

int main()
{
   int pid;
   pid=fork();
   if(pid<0)
   {
      printf("\n Error ");
      exit(1);
   }
   else if(pid==0)
   {
      printf("\n Hello I am the child process ");
      printf("\n My pid is %d ",getpid());
      exit(0);
   }
   else
   {
      printf("\n Hello I am the parent process ");
      printf("\n My actual pid is %d \n ",getpid());
      exit(1);
   }

}
```

I tried this , I hope its correct .
But I am not satisfied with the output .
The output is :

```
 Hello I am the parent process
 My actual pid is 4287
 ashu@ashu-VirtualWorld:~/Desktop/4thSemester/testprep$
 Hello I am the child process
 My pid is 4288
```

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/wait.h>
int main()
{
   int status;
   int pid;
   pid=fork();
   if(pid<0)
   {
      printf("\n Error ");
      exit(1);
   }
   else if(pid==0)
   {
      printf("\n Hello I am the child process ");
      printf("\n My pid is %d ",getpid());
      exit(0);
   }
   else
   {
     wait(&status);
      printf("\n Hello I am the parent process ");
      printf("\n My actual pid is %d \n ",getpid());
      exit(1);
   }

}
```

4. **Write a C program to simulate the following file organization techniques**
   **a) Single level directory**
   **b) Two level directory**
   **c) Hierarchical**

**DESCRIPTION:**

The directory structure is the organization of files into a hierarchy of folders. In a single-level directory system, all the files are placed in one directory. There is a root directory which has all files. It has a simple architecture and there are no sub directories. Advantage of single level directory system is that it is easy to find a file in the directory. In the two-level directory system, each user has own user file directory (UFD). The system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. When a user refers to a particular file, only his own UFD is searched. This effectively solves the name collision problem and isolates users from one another. Hierarchical directory structure allows users to create their own subdirectories and to organize their files accordingly. A tree is the most common directory structure. The tree has a root directory, and every file in the system has a unique path name. A directory (or subdirectory) contains a set of files or subdirectories.

**4a) Program for Single Level Directory**

```
#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\t5. Exit\n
        Enter 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
```

**4B) PROGRAM FOR TWO LEVEL 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].fname[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)
    printf("\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
En
ter 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
```

**4C) PROGRAM FOR HIERARCHICAL DIRECTORY ORGANIZATION**

```
#include<stdio.h>
#include<graphics.h>
 struct tree_element
{
        char name[20];
        int x, y, ftype, lx, rx, nc, level;
        struct tree_element *link[5];
};
typedef struct tree_element node; void main()
{
        int gd=DETECT,gm; node *root;
        root=NULL;
        clrscr();
        create(&root,0,"root",0,639,320);
        clrscr();
        initgraph(&gd,&gm,"c:\tc\BGI");
        display(root);
        getch();
        closegraph();
  }
  create(node **root,int lev,char *dname,int lx,int rx,int x)
  {
    int i, gap;
    if(*root==NULL)
    {
        (*root)=(node *)malloc(sizeof(node));
```

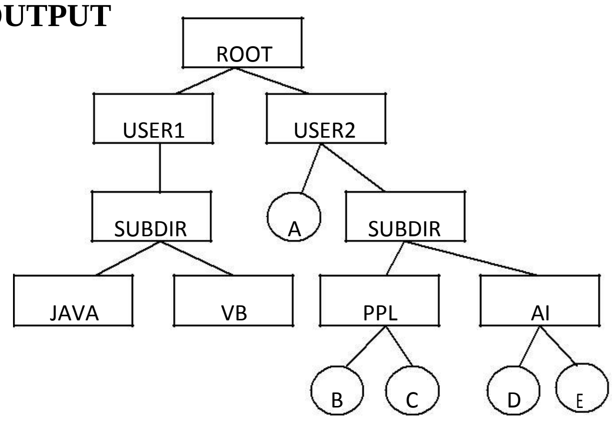
```
              printf("Enter name of dir/file(under %s) : ",dname);
              fflush(stdin);
             gets((*root)->name);
             printf("enter 1 for Dir/2 for file :");
             scanf("%d",&(*root)->ftype);
            (*root)->level=lev;
            (*root)->y=50+lev*50;
            (*root)->x=x;
            (*root)->lx=lx; (*root)->rx=rx;
             for(i=0;i<5;i++)
             (*root)->link[i]=NULL;
           if((*root)->ftype==1)
           {
               printf("No of sub directories/files(for %s):",(*root)->name);
               scanf("%d",&(*root)>nc);
              if((*root)->nc==0)
               gap=rx-lx;
            else
               gap=(rx-lx)/(*root)->nc;
               for(i=0;i<(*root)->nc;i++)
               create(&((*root)>link[i]),lev+1,(*root)>name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
        }
     else
            (*root)->nc=0;
      }
    }
    display(node *root)
    {
         int i;
         settextstyle(2,0,4);
         settextjustify(1,1);
         setfillstyle(1,BLUE);
         setcolor(14);
         if(root !=NULL)
          {
            for(i=0;i<root->nc;i++)
            line(root->x,root->y,root->link[i]->x,root->link[i]->y);
             if(root->ftype==1)
             bar3d(root->x-20,root->y-10,root->x+20,root>y+10,0,0);
             else
              fillellipse(root->x,root->y,20,20);
              outtextxy(root->x,root->y,root->name);
              for(i=0;i<root->nc;i++)
              display(root->link[i]);
        }
```

```
}
```

**INPUT**

```
Enter Name of dir/file(under root): ROOT
 Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for ROOT): 2
Enter Name of dir/file(under ROOT):
USER1 Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for USER1): 1
Enter Name of dir/file(under USER1):
SUBDIR1 Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for SUBDIR1): 2
Enter Name of dir/file(under USER1):
JAVA Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for JAVA): 0
Enter Name of dir/file(under SUBDIR1): VB
Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for VB): 0
Enter Name of dir/file(under ROOT):
USER2 Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for USER2):
2 Enter Name of dir/file(under ROOT):
A Enter 1 for Dir/2 for File: 2
Enter Name of dir/file(under USER2):
SUBDIR2 Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for SUBDIR2): 2
Enter Name of dir/file(under SUBDIR2):
PPL Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for PPL):
 2 Enter Name of dir/file(under PPL):
 B Enter 1 for Dir/2 for File: 2
Enter Name of dir/file(under PPL):
C Enter 1 for Dir/2 for File: 2
Enter Name of dir/file(under SUBDIR):
AI Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for AI):
2 Enter Name of dir/file(under AI):
D Enter 1 for Dir/2 for File: 2
Enter Name of dir/file(under AI):
E Enter 1 for Dir/2 for File: 2
```

**OUTPUT**



## 5. BANKERS ALGORITHM FOR THE PURPOSE OF DEADLOCK AVOIDANCE.

In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process. Banker’s algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type.

**PROGRAM**

```
#include<stdio.h>
          struct file
          {
                int all[10];
                int
                max[10];
                int
                need[10];
                int flag;
         };
       void main()
       {
                struct      file
                f[10]; int fl;
                int i, j, k, p, b, n, r, g, cnt=0,
                id, newr; int avail[10],seq[10];
                clrscr();
                printf("Enter number of processes -- ");
                scanf("%d",&n);
                printf("Enter number of resources -- ");
                scanf("%d",&r);
                for(i=0;i<n;i++)
                {
                          printf("Enter details for P%d",i);
                          printf("\nEnter allocation\t -- \t");
                          for(j=0;j<r;j++)
                           scanf("%d",&f[i].all[j]); printf("Enter Max\t\t -- \t");
                           for(j=0;j<r;j++)
                          scanf("%d",&f[i].max[j]);
                          f[i].flag=0;
                 }
                 printf("\nEnter Available Resources\t -- \t");
```

```
                    for(i=0;i<r;i++)
                    scanf("%d",&avail[i]);
                    printf("\nEnter New Request Details -- ");
                    printf("\nEnter pid \t -- \t");
                    scanf("%d",&id);
                    printf("Enter Request for Resources \t -- \t");
                    for(i=0;i<r;i++)
                    {
                    scanf("%d",&newr);
                    f[id].all[i] += newr;
                   avail[i]=avail[i] - newr;
       }
      for(i=0;i<n;i++)
        {
          for(j=0;j<r;j++)
           {
                     f[i].need[j]=f[i].max[j]-f[i].all[j];
                      if(f[i].need[j]<0)
                      f[i].need[j]=0;
           }
        }
   cnt=0;
   fl=0;
   while(cnt!=n)
   {
          g=0;
          for(j=0;j<n;j++)
          {
                 if(f[j].flag==0)
                 {
                        b=0;
                        for(p=0;p<r;p++)
                        {
                               if(avail[p]>=f[j].need[p])
                                      b=b+1;
                               else
                                      b=b-1;
                        }
                        if(b==r)
                        {
                              printf("\nP%d is visited",j); seq[fl++]=j;
                                f[j].flag=1;
                                for(k=0;k<r;k++)
                                avail[k]=avail[k]+f[j].all[k];
                                cnt=cnt+1;
                                printf("(");
                                for(k=0;k<r;k++)
                                printf("%3d",avail[k]);
                                printf(")");
                                g=1;
                        }
                        }
                }
                if(g==0)
                {
                        printf("\n REQUEST NOT GRANTED -- DEADLOCK OCCURRED");
                        printf("\n SYSTEM IS IN UNSAFE STATE");
                        goto y;
                }
        }
        printf("\nSYSTEM IS IN SAFE STATE");
        printf("\nThe Safe Sequence is -- (");
        for(i=0;i<fl;i++)
        printf("P%d ",seq[i]);
        printf(")");
      y:  printf("\nProcess\t\tAllocation\t\tMax\t\t\tNeed\n");
          for(i=0;i<n;i++)
           {
                printf("P%d\t",i);
                for(j=0;j<r;j++)
                      printf("%6d",f[i].all[j]);
                      for(j=0;j<r;j++)
                              printf("%6d",f[i].max[j]);
                      for(j=0;j<r;j++)
                              printf("%6d",f[i].need[j]);
                      printf("\n");
              }
              getch();
}
```

***INPUT***

```
Enter number of processes              –      5
Enter number of resources              --     3
Enter details for P0
Enter allocation                 --    0       1      0
Enter Max                      --            7      5      3

Enter details for P1
Enter allocation                 --    2      0      0
Enter Max                       --    3      2      2

Enter details for P2
Enter allocation                 --    3      0      2
Enter Max                       --    9      0      2


Enter details for P3
Enter allocation                 --    2      1      1
Enter Max                       --    2      2      2

Enter details for P4
Enter allocation                 --    0      0      2
Enter Max                       --    4      3      3
Enter Available Resources --                    3  3      2
Enter New Request Details --
Enter pid              --   1
Enter Request for Resources                     --  1        0     2
```

***OUTPUT***

```
P1 is visited       ( 5  3    2 )
P3 is visited       ( 7 4    3 )
P4 is visited       ( 7 4    5 )
P0 is visited       ( 7 5    5 )
P2 is visited       (10 5     7 )
SYSTEM IS IN SAFE STATE
The Safe Sequence is -- (P1 P3 P4 P0 P2 )

        Process         Allocation       Max          Need
        P0               0  1   0        7  5  3       7   4   3
        P1              3   0   2        3  2  2       0   2  0
        P2              3   0   2        9  0  2      6    0  0
        P3              2   1   1        2  2  2      0    1  1
        P4              0   0   2        4   3 3      4    3  1
```

## 6. PROGRAM FOR DEAD LOCK PREVENTION USING BANKERS ALGORITHM

```
#include< stdio.h>
#include< conio.h>
void main()
{
        Int allocated[15][15],max[15][15],need[15][15],avail[15],tres[15],work[15],flag[15];
         int  pno, rno,i,j,prc,count,t,total;
            count=0;
            clrscr();
            printf("\n Enter number of  process:");
            scanf("%d",&pno);
            printf("\n Enter number of resources:");
            scanf("%d",&rno);
            for(i=1;i< =pno;i++)
             {
                 flag[i]=0;
             }
             printf("\n Enter total numbers of each resources:");
             for(i=1;i<= rno;i++)
             scanf("%d",&tres[i]);
             printf("\n Enter Max resources for each process:");
            for(i=1;i<= pno;i++)
             {
               printf("\n for process %d:",i);
               for(j=1;j<= rno;j++)
               scanf("%d",&max[i][j]);
             }
             printf("\n Enter allocated resources for each process:");
             for(i=1;i<= pno;i++)
              {
                      printf("\n for process %d:",i);
                      for(j=1;j<= rno;j++)
                      scanf("%d",&allocated[i][j]);
               }
                     printf("\n available
                     resources:\n");
                      for(j=1;j<= rno;j++)
                      {
                              avail[j]=0;
                             total=0;
                              for(i=1;i<= pno;i++)
                             {
                                      total+=allocated[i][j];
                             }
                              avail[j]=tres[j]-total;
                             work[j]=avail[j];
                              printf("     %d\t",work[j]);
                      }
                       do
                      {
                              for(i=1;i<= pno;i++)
                              {
                                 for(j=1;j<= rno;j++)
                                      {
                                       need[i][j]=max[i][j]-allocated[i][j];
                                      }
```

```
          }
   printf("\n Allocated matrix        Max      need");
   for(i=1;i<= pno;i++)
    {
          printf("\n");
          for(j=1;j<=rno;j++)
           {
             printf("%4d",allocated[i][j]);
           }
          printf("|");
          for(j=1;j<= rno;j++)
            {
                  printf("%4d",max[i][j]);
           }
           printf("|");
          for(j=1;j<= rno;j++)
          {
             printf("%4d",need[i][j]);
          }
     }
    prc=0;
    for(i=1;i<= pno;i++)
    {
          if(flag[i]==0)
          {
                 prc=i;
                 for(j=1;j<= rno;j++)
                  {
                   if(work[j]< need[i][j])
                   {
                     prc=0;
                    break;
          }
     }
   }
   if(prc!=0)
        break;
}
   if(prc!=0)
   {
    printf("\n Process %d completed",i);
        count++;
    printf("\n Available
 matrix:");
    for(j=1;j<= rno;j++)
    {
      work[j]+=allocated[prc][j];
     allocated[prc][j]=0;
     max[prc][j]=0;
     flag[prc]=1;
     printf("   %d",work[j]);
    }
   }

  }
 while(count!=pno&&prc!=0);
   if(count==pno)
```

```
                printf("\nThe system is in a safe state!!");
                else
                printf("\nThe system is in an unsafe state!!");
                getch();
}

Enter number of process:5
Enter number of resources:3
Enter total numbers feachresources:10 5 7

Enter Max resources for each process:
 for process 1:7 5 3
 for process 2:3 2 2
 for process 3:9 0 2
 for process 4:2 2 2
 for process 5:4 3 3

 Enter allocated resources for each process:
 for process 1:0 1 0
 for process 2:3 0 2
 for process 3:3 0 2
 for process 4:2 1 1
 for process 5:0 0 2




 available resources:
    2     3     0
  Allocated matrix      Max        Need
       0  1  0          |  7  5  3  |  7  4  3
       3  0  2          |  3  2  2  |  0  2  0
       3  0  2          |  9  0  2  |  6  0  0
       2  1  1          |  2  2  2  |  0  1  1
       0  0  2          |  4  3  3  |  4  3  1
 Process 2 completed
 Available matrix:   5  3  2
 Allocated matrix      Max          Need
   0  1  0              |  7  5  3  |  7  4  3
   0  0  0              |  0  0  0  |  0  0  0
   3  0  2              |  9  0  2  |  6  0  0
   2  1  1              |  2  2  2  |  0  1  1
   0  0  2              |  4  3  3  |  4  3  1
 Process 4 completed
 Available matrix:   7  4  3
 Allocated matrix        Max           Need
      0  1  0            |  7  5  3  |     7  4  3
      0  0  0            |  0  0  0  |     0  0  0
      3  0  2            |  9  0  2  |     6  0  0
      0  0  0            |  0  0  0  |     0  0  0
      0  0  2            |  4  3  3  |     4  3  1
 Process 1 completed
 Available matrix:   7  5  3
 Allocated matrix      Max         need
  0  0  0             |  0  0  0  |  0  0  0
  0  0  0             |  0  0  0  |  0  0  0
  3  0  2             |  9  0  2  |  6  0  0
  0   0   0              |   0   0   0   |   0   0   0
  0   0   2              |   4   3   3   |   4   3   1
 Process 3 completed
 Available matrix:   10   5   5
 Allocated matrix           Max      need
  0   0   0            |   0   0   0      |   0   0   0
  0   0   0            |   0   0   0      |   0   0   0
  0   0   0            |   0   0   0      |   0   0   0
  0   0   0            |   0   0   0      |   0   0   0
  0   0   2            |   4   3   3      |   4   3   1
 Process 5 completed
 Available matrix:   10   5   7
The system is in a safe state!!
```

**7. Write a C program to simulate page replacement algorithms**
**a) FIFO  b) LRU  c) LFU**

DESCRIPTION
Page replacement is basic to demand paging. It completes the separation between logical memory and physical memory. With this mechanism, an enormous virtual memory can be provided for programmers on a smaller physical memory. There are many different page-replacement algorithms. Every operating system probably has its own replacement scheme. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. If the recent past is used as an approximation of the near future, then the page that has not been used for the longest period of time can be replaced. This approach is the Least Recently Used (LRU) algorithm. LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses the page that has not been used for the longest period of time. Least frequently used (LFU) page-replacement algorithm requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count.

**7A) Program for FIFO PAGE REPLACEMENT ALGORITHM**

```
#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();
}
```

**INPUT**

```
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
```

**OUTPUT**

```
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
```

**7b). LRU PAGE REPLACEMENT PROGRAM**

```
#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();
}
```

**INPUT**

```
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
```

***OUTPUT***

```
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
```

## 7c) LFU PAGE REPLACEMENT PROGRAM

```
        #include<stdio.h>
        #include<conio.h>
        main()
        {
               int rs[50], i, j, k, m, f, cntr[20], a[20], min, pf=0;
               clrscr();
               printf("\nEnter number of page references -- ");
               scanf("%d",&m);
               printf("\nEnter the reference string -- ");
              for(i=0;i<m;i++)
              scanf("%d",&rs[i]);
              printf("\nEnter the available no. of frames -- ");
              scanf("%d",&f);
        for(i=0;i<f;i++)
        {
                   cntr[i]=0; a[i]=-1;
        }
        Printf("\nThe Page Replacement Process is – \n");
        for(i=0;i<m;i++)
        {
          for(j=0;j<f;j++)
          if(rs[i]==a[j])
          {
                   cntr[j]++;
                   break;
         }
        if(j==f)
       {
        min = 0;
        for(k=1;k<f;k++)
        if(cntr[k]<cntr[min])
        min=k;
        a[min]=rs[i];
        cntr[min]=1;
        pf++;
        }
        printf("\n");
  for(j=0;j<f;j++)
    printf("\t%d",a[j]);
    if(j==f)
  printf("\tPF No. %d",pf);
  }
printf("\n\n Total number of page faults -- %d",pf);
```

```
  getch();
}
```

**INPUT**

```
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
```

**OUTPUT**

```
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
```

8. **PROGRAM FOR SHARE MEMORY AND IPC**

```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/sem.h> int main()
{
     int id,semid,count=0,i=1,j;
     int *ptr;
     id=shmget(8078,sizeof(int),IPC_CREAT|0666);
     ptr=(int *)shmat(id,NULL,0);
   union semun
   {
      int val;
      struct semid_ds *buf; ushort *array;
   }u;
 struct sembuf sem; semid=semget(1011,1,IPC_CREAT|0666);
 ushort a[1]={1};
 u.array=a;
 semctl(semid,0,SETALL,u);
 while(1)
{
  sem.sem_num=0;
  sem.sem_op=-1;
  sem.sem_flg=0;
  semop(semid,&sem,1);
  *ptr=*ptr+1;
  printf("process id:%d countis :%d \n",getpid(),*ptr);  for(j=1;j<=1000000;j++)
 {
    sem.sem_num=0;
    sem.sem_op=+1;
    sem.sem_flg=0;
    semop(semid,&sem,1);
 }
}
  shmdt(ptr);
```

**RESULT:**

Thus the program was executed