JMIETI, RADAUR

PRACTICAL FILE
OF
"OPERATING SYSTEM"
(PC-CS212L)

Submitted To:-   **Submitted By:-**

Mr. Sumit Kumar Mahana   **Mridul**

Assistant Professor   **8521174**

(CSE Department)   **(CSE – 4th Sem)**

Jai Prakash Mukand Lal Innovative Engineering & Technology Institute (JMIETI)

Affilated To Kurukshetra University, Kurukshetra – Haryana, Radaur-135133

Mridul (8521174)

<u>INDEX</u>

Mridul (8521174)

PROGRAM-1

AIM: To Study the Hardware and Software Requirements of various operating systems.

Operating Systems:

An operating system (OS) is system software that manages computer hardware and software resources and provides common services for computer programs. All computer programs, excluding firmware, require an operating system to function.

Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources.

Functions of Operating Systems:

- ➢ Storage Management
- ➢ Process Management
- ➢ Extended Machine
- ➢ Mastermind
- ➢ Processor Management
- ➢ Device Management
- ➢ Security
- ➢ Accounting

Types of Operating Systems:

- ➢ MS-DOS
- ➢ Windows
- ➢ Linux
- ➢ Unix
- ➢ IOS

MS-DOS:

DOS stands for **disk operating system**, is an acronym for several computer operating systems that are operated by using the command line.

MS-DOS dominated the IBM PC compatible (PC and PC Compatible) market between 1981 and 1995, or until about 2001 including the partially MS-DOS based Microsoft Windows (95, 98, and Millennium Edition). "DOS" is used to describe the family of several very similar command-line systems, including MS-DOS, PC-DOS, DR-DOS, Free-DOS, ROM-DOS, and PTS-DOS.

FEATURES:

 ➢ It is a single user operating system and is a command user interface (CUI). It interprets command typed on the DOS mode.
 ➢ The Microsoft Anti-virus command is a program that can identify and remove more than 800 different computer viruses from user system.
 ➢ Microsoft UNDELETE command id quite useful in recovering all the deleted files.

System Requirements for MS-DOS:

 Hardware Requirements:

 ➢ At least a 486DX33 with 16MB RAM is required. A Pentium Pro and more main memory are recommended. A 386 or a system with 8MB or less memory is an insufficient configuration.
 ➢ There are no specific requirements concerning network cards, disk types, or CD-ROM equipments; of course the more powerful, the better.
 ➢ Depending on the packages installed, a disk space of 20-55MB on a HPFS formatted partition (or a JFS, NFS or ext2fs partition natively allowing long filenames) is required. XFree86/OS2 will not run on FAT partitions.

 Software Requirements:

 ➢ Any version of Warp 3 with at least fixpack17 or Warp 4 is required.
 ➢ XFree86/OS2-3.3.6 may use a local named-pipe connection or a TCP/IP based network connection.
 ➢ Warp comes with the Internet Access Kit (IAK), which is sufficient. Warp Connect and Warp Server come with a full version of TCP/IP (3.0). Use of this software is preferred over IAK.

Windows:

Windows 8 is a <u>personal computer operating system</u> developed by Microsoft as a part of the <u>Windows NT</u> family of operating systems. Development of Windows 8 started before the release of its predecessor, <u>Windows 7</u>, in 2009. It was announced at <u>CES</u> 2011 to May 2012. The operating system was <u>released to manufacturing</u> on August1, 2012, and was released for <u>general availability</u> on October26, 2012.

Features:

New features and functionality in Windows 8 include a faster startup through <u>UEFI</u> integration and the new "Hybrid Boot" mode(which <u>hibernates</u> the Window <u>Kernel</u> on shutdown to speed up the subsequent boot), a new lock screen with a clock and notifications, and the ability for enterprise users to create <u>live USB</u> versions of Windows(known as <u>Windows To Go</u>). Windows 8 also adds native support for <u>USB 3.0</u> devices, which allow for faster data transfers and improved power management with compatible devices, and <u>hard disk</u> 4 KB <u>Advanced Formal</u> support, as well as support for <u>near field communication</u> to facilitate sharing and communication between devices.

Hardware Requirements:

Processor	1 GHz clock rate IA-32 or x64 architecture support for PAE, NX and SSE2.	x64 architecture Second Level Address Translation (SLAT) support for Hyper-V.
Memory (RAM)	IA-32 edition: 1 GB x64 edition: 2GB	4GB
Graphics Card	DirectX 9 graphics device WDDM 1.0 or higher driver	DirectX 10 graphics device

Software Requirements:

The three desktop editions of Windows8 support 32-bit and 64-bit architectures; retail copies of Windows8 include installs DVDs for both architectures, while the architecture of the system's existing Windows installation. The 32-bit version runs on CPUs compatible with <u>x86</u> architecture 3rd <u>generation</u> (known as <u>IA-32</u>) or newer, and can run <u>32-bit</u> and <u>16-bit</u> <u>applications</u>, although 16-bit support must be enabled first. (16-bit applications are developed for CPUs compatible with <u>x86</u> 2nd<u>generation, first conceived in 1978</u>. Microsoft started moving away from this architecture after <u>Windows95</u>.

<u>Linux:</u>

Mridul (8521174)

Linux or, less frequently, is a <u>Unix-like</u> and mostly <u>POSIX</u>-compliant computer <u>operating system</u> (OS) assembled under the model of <u>free and open-source software</u> development and distribution. The defining component of Linux is the <u>Linux Kernel</u>, an <u>operating system kernel</u> first released on October5, 1991 by <u>Linus Torvalds</u>. The <u>Free Software Foundation</u> uses the name GNU/Linux to describe the operating system, which has led to some <u>controversy</u>.

Features:

- ➤ **Portable-** Portability means software's can works on different types of hardware's in same way .Linux kernel and application programs supports their installation on any kind of hardware platform.
- ➤ **Open Source-** Linux source code is freely available and it is community based development project. Multiple-team works in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- ➤ **Multi-User-** Linux is a multiuser system means multiple users can access system resources like memory/RAM/application programs at same time.

Hardware Requirements:

Criteria	Requirements
Operating System	Red Hat Enterprise Linux 4 or 5 with the latest patches.
CPU Type	Pentium 4 or higher; 2 GHz or higher.
Memory/RAM	1 GB minimum, up to the system limit.

Software Requirements:

- ➤ gcc-3.3.3-43.24
- ➤ gcc-c++-3.3.3-43.24
- ➤ libstdc++-3.3.3-43.24
- ➤ pdksh-5.2.14-780.1
- ➤ gnome-libs-1.4.1.7-671.1

UNIX:

Mridul (8521174)

UNIX trademarked as **UNIX** is a family of <u>multitasking</u>, <u>multiuser</u> computer <u>operating systems</u> that derive from the original <u>AT&T</u> UNIX, developed in the 1970s at the <u>Bell Labs</u> research center by <u>Ken Thompson</u>, <u>Dennis Ritchie</u>, and others.

Initially intended for use inside the <u>Bell System</u>, AT&T licensed Unix to outside parties from the late 1970s, leading to a variety of both academic and commercial variants of Unix from vendors such as the <u>University of California</u>, <u>Berkeley (BSD)</u>, <u>Microsoft (Xenix)</u>, <u>IBM (AIX)</u> and <u>Sun Microsystems (Solaris)</u>. AT&T finally sold its right in Unix to <u>Novell</u> in early 1990s, which then sold its Unix business to the <u>Santa Cruz Operation</u> (SCO) in 1995, but the UNIX trademark passed to the industry standards consortium <u>The Open Group</u>, which allows the use of the mark for certified operating systems compliant with the <u>Single UNIX Specifications</u> (SUS). Among these is <u>Apple's OS X</u>, which is the Unix version with the largest installed base as of 2014.

Features:

- ➢ It is a good OS, especially, for programs. UNIX programming environment is unusually rich and productive. It provides features that allow complex programs to be built from simpler programs.
- ➢ It uses a hierarchical file system that allows easy maintenance and efficient implementation.

Hardware Requirements:

Item	HP 9000 (8xx)	RISC/6000 AIX
CPU	HP 9000/800	IBM RISC System/6000
RAM	256 MB, minimum.	256MB, minimum.
Storage	Minimum of 300MB, and at least 500KB for each local you support.	Minimum of 300MB, and at least 500KB for each local you support.

Software Requirements:

Platform requirements	HP-UX 11.23	AIX 5.3	Sun Solaris 2.9 and 2.10	Linux Red Hat3.0,4.0, and5.0 SuSE9.0 and 10	HP-UX 11.31 ia64
ODBC driver	ODBS 3.5-compliant driver to access the targeted database, on the same platform where you are using ECDA for ODBC (DB2 UDB Target DBMS).				

IOS:

Mridul (8521174)

Mac OS X Lion (version 10.7; marketed as **OS X Lion**) is the eighth <u>major release</u> of <u>OS X</u>, <u>Apple's</u> <u>desktop</u> and server <u>operating system</u> for <u>Macintosh</u> computers.

A preview of Lion was publicly unveiled at the "Back to the Mac" Apple Special Event on October20, 2010. It brings many developments made in Apple's iOS, such as an easily navigable display of installed applications, to the Mac, and includes support for the Mac App Store, as introduced in Mac OS X Snow Leopard version 10.6.6. On February 24, 2011, the first developer's preview of Lion (11A390) was released to subscribers to the Apple Developer program. Other developer previews were subsequently released, with Lion Preview 4 (11A480b) being released at WWDC 2011.

Features:

- ➤ **Battery usage indicators-** It's probably the most common complaint about the iPhone. The battery drains too damn fast. This is, of course, dependent on exactly what you do with it, but how do users know whether to prioritize the screen.
- ➤ **New Keyboard-** There's nothing more fundamental to the iPhone than its onscreen keyboard-and in iOS 8, it gets an upgrade via predictive typing, which suggests several options for the next word as you type. This is a feature that's been on Android for a while, although Apple says its implementation is superior-since it learns what you're likely to say to different friends and colleagues.

Continuity- Apple took the idea of collaboration through the cloud to a new level with a new feature called Handoff, part of its "Community" concept. If you have an iPhone and a Mac, you'll be able to start a task on one device (say, composing an email) and finish on the other. Since the devices are aware of each other, all you have to do is click one button, and it works on iPad, too.

Hardware Requirements:

- ➤ x86-64 CPU (64-bit Macs, with an Intel Core 2 Duo, Intel Core i3, Intel Core i5, Intel Core i7, or Xeon processor).
- ➤ At least 2 GB of Ram.
- ➤ Mac OS X 10.6.6 or later (Mac OS X 10.6.8 is recommended).

Software Requirements:

- ➤ Applications depending on Rosetta, such as Office for Mac 2004, AppleWorks, and early versions of Quicken for Mac 2007, are no longer supported. This affects applications listed as *Classic* or *Power PC* in System Profiler.
- ➤ Unix package managers for Mac OS X such as Fink and Mac Ports require reinstalling and then running Xcode.

PROGRAM-2

Aim: C++ program for implementation of FCFS scheduling

SOURCE CODE:

```cpp
#include<iostream>

using namespace std;

void findWaitingTime(int processes[], int n,

                int bt[], int wt[])

{

    wt[0] = 0;

    for (int  i = 1; i < n ; i++ )

        wt[i] =  bt[i-1] + wt[i-1] ;

}

void findTurnAroundTime( int processes[], int n,

            int bt[], int wt[], int tat[])

{

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

        tat[i] = bt[i] + wt[i];

}

void findavgTime( int processes[], int n, int bt[])

{

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

    findWaitingTime(processes, n, bt, wt);

    findTurnAroundTime(processes, n, bt, wt, tat);
```

Mridul (8521174)

```cpp
        cout << "Processes  "<< " Burst time  "

          << " Waiting time  " << " Turn around time\n";

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

     {

        total_wt = total_wt + wt[i];

        total_tat = total_tat + tat[i];

        cout << "   " << i+1 << "\t\t" << bt[i] <<"\t   "

           << wt[i] <<"\t\t  " << tat[i] <<endl;

     }

     cout << "Average waiting time = "

        << (float)total_wt / (float)n;

     cout << "\nAverage turn around time = "

        << (float)total_tat / (float)n;

}

int main()

{

     int processes[] = { 1, 2, 3};

    int n = sizeof processes / sizeof processes[0];

    int  burst_time[] = {10, 5, 8};

    findavgTime(processes, n,  burst_time);

    return 0;

}
```

OUTPUT:

```
Processes    Burst time    Waiting time    Turn around time
    1            10            0                10
    2            5             10               15
    3            8             15               23
Average waiting time = 8.33333
Average turn around time = 16

...Program finished with exit code 0
Press ENTER to exit console.
```

PROGRAM-3

Aim: C++ program to implement Shortest Job first with Arrival Time

SOURCE CODE:

```cpp
#include<iostream>

using namespace std;
int main()
{
    int n,temp,tt=0,min,d,i,j;
    float atat=0,awt=0,stat=0,swt=0;

    cout<<"enter no of process"<<endl;
    cin>>n;
    int a[n],b[n],e[n],tat[n],wt[n];

    for(i=0;i<n;i++)
    {
        cout<<"enter arival time ";
        cin>>a[i];
    }
    for(i=0;i<n;i++)
    {
        cout<<"enter brust time ";
        cin>>b[i];
    }
    for(i=0;i<n;i++)
    {
      for(j=i+1;j<n;j++)
       {
           if(b[i]>b[j])
           {
               temp=a[i];
               a[i]=a[j];
               a[j]=temp;
```

```
                temp=b[i];
                b[i]=b[j];
                b[j]=temp;
            }
        }
    }
    min=a[0];
    for(i=0;i<n;i++)
    {
        if(min>a[i])
        {
            min=a[i];
            d=i;
        }
    }
    tt=min;
    e[d]=tt+b[d];
    tt=e[d];

    for(i=0;i<n;i++)
    {
        if(a[i]!=min)
        {
            e[i]=b[i]+tt;
            tt=e[i];
        }
    }
    for(i=0;i<n;i++)
    {

        tat[i]=e[i]-a[i];
        stat=stat+tat[i];
        wt[i]=tat[i]-b[i];
        swt=swt+wt[i];
    }
    atat=stat/n;
    awt=swt/n;
    cout<<"Process  Arrival-time(s)  Burst-time(s)  Waiting-time(s)  Turnaround-ti
```

```cpp
me(s)\n";

    for(i=0;i<n;i++)
    {
    cout<<"P"<<i+1<<"           "<<a[i]<<"           "<<b[i]<<"           "<<wt
[i]<<"           "<<tat[i]<<endl;
    }

    cout<<"awt="<<awt<<" atat="<<atat;  //average waiting time and turn around ti
me
}
```

OUTPUT:

```
enter no of process
5
enter arival time 3
enter arival time 1
enter arival time 4
enter arival time 0
enter arival time 2
enter brust time 1
enter brust time 4
enter brust time 2
enter brust time 6
enter brust time 3
Process  Arrival-time(s)  Burst-time(s)  Waiting-time(s)  Turnaround-time(s)
P1            3               1              3               4
P2            4               2              3               5
P3            2               3              7               10
P4            1               4              11               15
P5            0               6              0               6
awt=4.8 atat=8

...Program finished with exit code 0
Press ENTER to exit console.
```

PROGRAM-4

Aim: Implementing Priority Scheduling Algorithm in C++.

SOURCE CODE:

```cpp
#include<iostream>

using namespace std;
int main()
{
    int a[10],b[10],x[10];
    int waiting[10],turnaround[10],completion[10],p[10];
    int i,j,smallest,count=0,time,n;
    double avg=0,tt=0,end;

    cout<<"\nEnter the number of Processes: ";
    cin>>n;
    for(i=0;i<n;i++)
    {
        cout<<"\nEnter arrival time of process: ";
        cin>>a[i];
    }
    for(i=0;i<n;i++)
    {
        cout<<"\nEnter burst time of process: ";
        cin>>b[i];
    }
    for(i=0;i<n;i++)
    {
        cout<<"\nEnter priority of process: ";
        cin>>p[i];
    }
    for(i=0; i<n; i++)
        x[i]=b[i];

    p[9]=-1;
```

```cpp
    for(time=0; count!=n; time++)
    {
        smallest=9;
        for(i=0; i<n; i++)
        {
            if(a[i]<=time && p[i]>p[smallest] && b[i]>0 )
                smallest=i;
        }
        b[smallest]--;

        if(b[smallest]==0)
        {
            count++;
            end=time+1;
            completion[smallest] = end;
            waiting[smallest] = end - a[smallest] - x[smallest];
            turnaround[smallest] = end - a[smallest];
        }
    }
    cout<<"Process"<<"\t"<< "burst-time"<<"\t"<<"arrival-time" <<"\t"<<"waiting-time" <<"\t"<<"turnaround-time"<< "\t"<<"completion-time"<<"\t"<<"Priority"<<endl;
    for(i=0; i<n; i++)
    {
        cout<<"p"<<i+1<<"\t\t"<<x[i]<<"\t\t"<<a[i]<<"\t\t"<<waiting[i]<<"\t\t"<<turnaround[i]<<"\t\t"<<completion[i]<<"\t\t"<<p[i]<<endl;
        avg = avg + waiting[i];
        tt = tt + turnaround[i];
    }
    cout<<"\n\nAverage waiting time ="<<avg/n;
    cout<<"  Average Turnaround time ="<<tt/n<<endl;
}
```

OUTPUT:

```
Enter the number of Processes: 4

Enter arrival time of process: 1

Enter arrival time of process: 3

Enter arrival time of process: 2

Enter arrival time of process: 5

Enter burst time of process: 3

Enter burst time of process: 6

Enter burst time of process: 7

Enter burst time of process: 8

Enter priority of process: 2

Enter priority of process: 4

Enter priority of process: 1

Enter priority of process: 3
Process burst-time     arrival-time    waiting-time    turnaround-time completion-time Priority
p1           3              1               14              17              18              2
p2           6              3               0               6               9               4
p3           7              2               16              23              25              1
p4           8              5               4               12              17              3


Average waiting time =8.5  Average Turnaround time =14.5
```

Mridul (8521174)

PROGRAM-05

Aim: C Program to implement Round Robin Scheduling Algorithm.

SOURCE CODE:

```
1.  #include<stdio.h>
2.  #include<conio.h>
3.
4.  void main()
5.  {
6.      int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
7.      float avg_wt, avg_tat;
8.      printf(" Total number of process in the system: ");
9.      scanf("%d", &NOP);
10.     y = NOP; // Assign the number of process to variable y
11. for(i=0; i<NOP; i++)
12. {
13. printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
14. printf(" Arrival time is: \t");
15. scanf("%d", &at[i]);
16. printf(" \nBurst time is: \t");
17. scanf("%d", &bt[i]);
18. temp[i] = bt[i];
19. }
20. printf("Enter the Time Quantum for the process: \t");
21. scanf("%d", &quant);
22. printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
23. for(sum=0, i = 0; y!=0; )
24. {
25. if(temp[i] <= quant && temp[i] > 0) // define the conditions
26. {
27.     sum = sum + temp[i];
```

```
28.    temp[i] = 0;
29.    count=1;
30.    }
31.    else if(temp[i] > 0)
32.    {
33.       temp[i] = temp[i] - quant;
34.       sum = sum + quant;
35.    }
36.    if(temp[i]==0 && count==1)
37.    {
38.       y--;
39.       printf("\nProcess No[%d] \t\t %d\t\t\t %d\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);
40.       wt = wt+sum-at[i]-bt[i];
41.       tat = tat+sum-at[i];
42.       count =0;
43.    }
44.    if(i==NOP-1)
45.    {
46.       i=0;
47.    }
48.    else if(at[i+1]<=sum)
49.    {
50.       i++;
51.    }
52.    else
53.    {
54.       i=0;
55.    }
56. }
57. avg_wt = wt * 1.0/NOP;
58. avg_tat = tat * 1.0/NOP;
59. printf("\n Average Turn Around Time: \t%f", avg_wt);
60. printf("\n Average Waiting Time: \t%f", avg_tat);
```

Mridul (8521174)

61. getch();
62. }

OUTPUT:

```
 Total number of process in the system: 4

 Enter the Arrival and Burst time of the Process[1]
 Arrival time is:        0

Burst time is:  8

 Enter the Arrival and Burst time of the Process[2]
 Arrival time is:        1

Burst time is:  5

 Enter the Arrival and Burst time of the Process[3]
 Arrival time is:        2

Burst time is:  10

 Enter the Arrival and Burst time of the Process[4]
 Arrival time is:        3

Burst time is:  11
Enter the Time Quantum for the process:            6

 Process No                 Burst Time              TAT              Waiting Time

 Process No[2]              5                10              5
 Process No[1]              8                25              17
 Process No[3]              10               27              17
 Process No[4]              11               31              20
 Average Turn Around Time:      14.750000
 Average Waiting Time:  23.250000
```

Mridul (8521174)

Program – 06

AIM: **Implementing Banker's Algorithm in C++.**

Source Code:

```cpp
#include <iostream>

using namespace std;

int main()

{

  int n, m, i, j, k;

  n = 5;

  m = 3;

  int alloc[5][3] = { { 0, 1, 0 },

            { 2, 0, 0 }

            { 3, 0, 2 }

            { 2, 1, 1 }

            { 0, 0, 2 } }

  int max[5][3] = { { 7, 5, 3 }

         { 3, 2, 2 }

         { 9, 0, 2 }

         { 2, 2, 2 }

         { 4, 3, 3 } }

  int avail[3] = { 3, 3, 2 };

  int f[n], ans[n], ind = 0;

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

{
```

```c
        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];
```
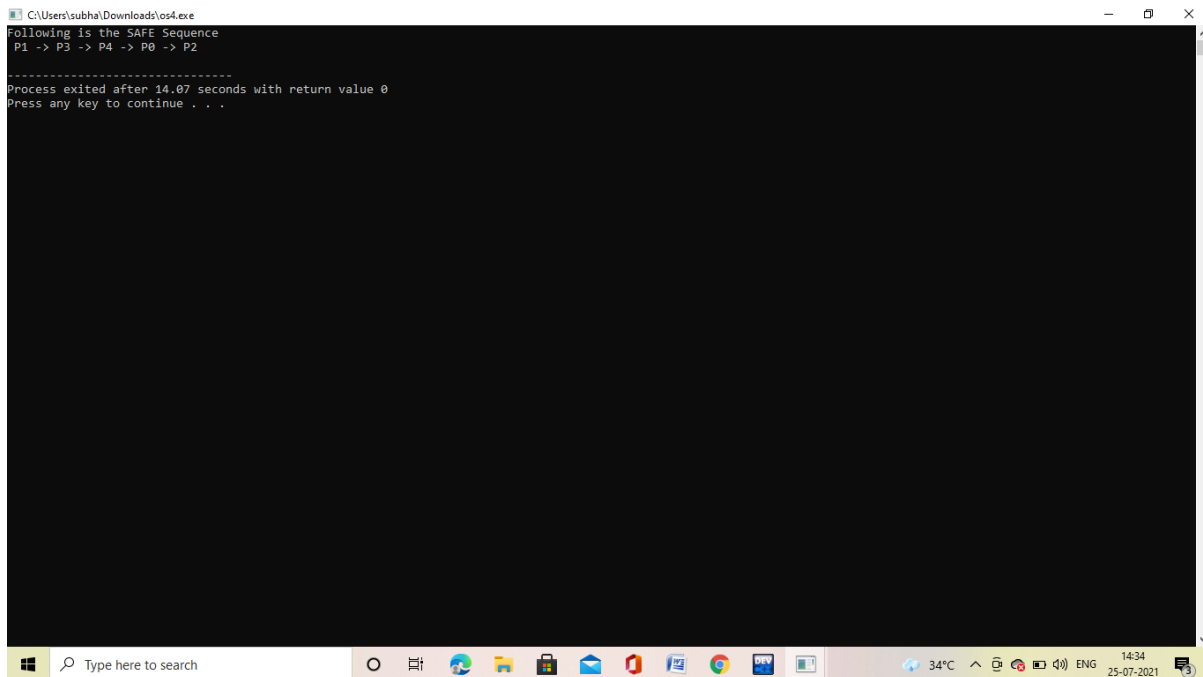
```cpp
                f[i] = 1;
            }
        }
    }

    cout << "Following is the SAFE Sequence" << endl;
    for (i = 0; i < n - 1; i++)
        cout << " P" << ans[i] << " ->";
    cout << " P" << ans[n - 1] <<endl;
    return (0);
}
```

Output:

Mridul (8521174)

Program – 07

AIM: C++ implementation of First-fit algorithm.

Source Code:

```cpp
#include<bits/stdc++.h>

using namespace std;

void firstFit(int blockSize[], int m,

                int processSize[], int n)

{

        int allocation[n];

        memset(allocation, -1, sizeof(allocation));

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

    {

        for (int j = 0; j < m; j++)

        {

            if (blockSize[j] >= processSize[i])

            {

                allocation[i] = j;

                blockSize[j] -= processSize[i];

                break;

            }

        }

    }


    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
```

Mridul (8521174)

```cpp
        for (int i = 0; i < n; i++)

        {

                cout << " " << i+1 << "\t\t"

                        << processSize[i] << "\t\t";

                if (allocation[i] != -1)

                        cout << allocation[i] + 1;

                else

                        cout << "Not Allocated";

                cout << endl;

        }

}

int main()

{

        int blockSize[] = {100, 500, 200, 300, 600};

        int processSize[] = {212, 417, 112, 426};

        int m = sizeof(blockSize) / sizeof(blockSize[0]);

        int n = sizeof(processSize) / sizeof(processSize[0]);

        firstFit(blockSize, m, processSize, n);

        return 0 ;

}
```

Output:

```
C:\Users\subha\Downloads\os7.exe                                                    —    □    ×

Enter the total no. of processes: 7

Enter size of each process: Process No.[1]:    25
Process No.[2]: 300
Process No.[3]: 50
Process No.[4]: 120
Process No.[5]: 25
Process No.[6]: 12
Process No.[7]: 10

Process No.      Process Size      Block no.

1                25                1

2                300               5

3                50                1

4                120               2

5                25                1

6                12                2

7                10                2

----------------------------------
Process exited after 39.61 seconds with return value 0
Press any key to continue . . .
```

Program – 08

AIM: C++ implementation of Best-fit algorithm.

Source Code:

```cpp
#include<iostream>
using namespace std;
int main()
{
int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
static int barray[20],parray[20];
cout<<"\n\t\t\tMemory Management Scheme - Best Fit";
cout<<"\nEnter the number of blocks:";
cin>>nb;
cout<<"Enter the number of processes:";
cin>>np;
cout<<"\nEnter the size of the blocks:-\n";
for(i=1;i<=nb;i++)
   {
      cout<<"Block no."<<i<<":";
      cin>>b[i];
   }
cout<<"\nEnter the size of the processes :-\n";
for(i=1;i<=np;i++)
   {
      cout<<"Process no. "<<i<<":";
      cin>>p[i];
   }
for(i=1;i<=np;i++)
{
for(j=1;j<=nb;j++)
{
if(barray[j]!=1)
{
temp=b[j]-p[i];
if(temp>=0)
if(lowest>temp)
{
parray[i]=j;
lowest=temp;
}
}
}
fragment[i]=lowest;
```

Mridul (8521174)

```cpp
barray[parray[i]]=1;
lowest=10000;
}
cout<<"\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment";
for(i=1;i<=np && parray[i]!=0;i++)
cout<<"\n"<<i<<"\t\t"<<p[i]<<"\t\t"<<parray[i]<<"\t\t"<<b[parray[i]]<<"\t\t"<<fragment[i];
return 0;
}
```

Output:

```
C:\Users\subha\Downloads\os6.exe                                                    —  □  ×

Enter the Total Number of Blocks:      5

Enter the Total Number of Processes:    7

Enter the Size of the Blocks :  Block No.[1]:   100
Block No.[2]:    50
Block No.[3]:    250
Block No.[4]:    400
Block No.[5]:    230
Enter the Size of the Process:
Process No.[1]: 200
Process No.[2]: 50
Process No.[3]: 30
Process No.[4]: 10
Process No.[5]: 300
Process No.[6]: 100
Process No.[7]: 50
1               200             5
2               50              2
3               30              5
4               10              1
5               300             4
6               100             4
7               50              1

--------------------------------
Process exited after 61.73 seconds with return value 0
Press any key to continue . . .
```

Program – 09

AIM: C++ implementation of Worst-fit algorithm.

Source Code:

```cpp
#include<bits/stdc++.h>
using namespace std;
void worstFit(int blockSize[], int m, int processSize[],
                              int n)
{

    int allocation[n];

    memset(allocation, -1, sizeof(allocation));

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

        int wstIdx = -1;
        for (int j=0; j<m; j++)
        {
            if (blockSize[j] >= processSize[i])
            {
                if (wstIdx == -1)
                    wstIdx = j;
                else if (blockSize[wstIdx] < blockSize[j])
                    wstIdx = j;
            }
        }

        // If we could find a block for current process
        if (wstIdx != -1)
        {

            allocation[i] = wstIdx;

            blockSize[wstIdx] -= processSize[i];
        }
    }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
    for (int i = 0; i < n; i++)
    {
        cout << "   " << i+1 << "\t\t" << processSize[i] << "\t\t";
```

Mridul (8521174)

```cpp
        if (allocation[i] != -1)
            cout << allocation[i] + 1;
        else
            cout << "Not Allocated";
        cout << endl;
    }
}

int main()
{
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize)/sizeof(blockSize[0]);
    int n = sizeof(processSize)/sizeof(processSize[0]);

    worstFit(blockSize, m, processSize, n);

    return 0 ;
}
```

Output:

```
C:\Users\subha\Downloads\os8.exe

Enter the Total Number of Blocks:       5

Enter the Total Number of Processes:    6

Enter the Size of the Blocks:
Block No.[1]:    300
Block No.[2]:    200
Block No.[3]:    100
Block No.[4]:    250
Block No.[5]:    50
Enter the Size of the Process:
Process No.[1]: 80
Process No.[2]: 100
Process No.[3]: 300
Process No.[4]: 250
Process No.[5]: 50
Process No.[6]: 10

Process No.      Process Size    Block no.
1                80              1
2                100             4
3                300             Not Allocated
4                250             Not Allocated
5                50              1
6                10              2

--------------------------------
Process exited after 60.88 seconds with return value 0
Press any key to continue . . .
```

Program – 10

AIM: C++ implementation of Internal fragmentation.

Source Code:

```cpp
#include<iostream>
#include<algorithm>
using namespace std;
struct node{
    int memsize;
    int allocp=-1;
    int pos;
    int allocSize;
}  m[200];
bool posSort(node a,node b){
    return a.pos < b.pos;
}
bool memSort(node a,node b){
    return a.memsize < b.memsize;
}
int main()
{
    int nm,np,choice, i, j, p[200];
    cout<<"Enter number of blocks\n";
    cin>>nm;
    cout<<"Enter block size\n";
```

```cpp
    for(i=0;i<nm;i++){

        cin>>m[i].memsize;

        m[i].pos=i;   }

    cout<<"Enter number of processes\n";

    cin>>np;

cout<<"Enter process size\n";

    for(i=0;i<np;i++){

        cin>>p[i];

    }

    cout<<"\n\n";

    sort(m,m+nm,memSort);

    int globalFlag=0;

    for(i=0;i<np;i++){

        int flag=0;

        for(j=0;j<nm;j++){

            if(p[i]<=m[j].memsize && m[j].allocp==-1){

                m[j].allocp=i;

                m[j].allocSize=p[i];

                flag=1;

                break;

            }

        }

        if(flag==0){

            cout<<"Unallocated Process P"<<i+1<<"\n";
```

```cpp
            globalFlag=1;

        }

    }

sort(m,m+nm,posSort);

cout<<"\n";

int intFrag=0,extFrag=0;

cout<<"Memory\t\t";

for(i=0;i<nm;i++){

    cout<<m[i].memsize<<"\t";

}

cout<<"\n";

cout<<"P. Alloc.\t";

for(i=0;i<nm;i++){

    if(m[i].allocp!=-1){

        cout<<"P"<<m[i].allocp+1<<"\t";

    }

    else{

        cout<<"Empty\t";

    }

}

cout<<"\n";

cout<<"Int. Frag.\t";

for(i=0;i<nm;i++){

        if(m[i].allocp!=-1){
```

```cpp
            cout<<m[i].memsize-m[i].allocSize<<"\t";

            intFrag+=m[i].memsize-m[i].allocSize;      }

        else{

            extFrag+=m[i].memsize;

            cout<<"Empty\t";

        }    }

    cout<<"\n";

    cout<<"\n";

    if(globalFlag==1)

        cout<<"Total External Fragmentation: "<<extFrag<<"\n";

    else

    {    cout<<"Available Memory: "<<extFrag<<"\n";    }

cout<<"Total Internal Fragmentation: "<<intFrag<<"\n";

    return 0;

}
```

Output:

```
Enter number of blocks
4
Enter block size
15
17
23
27
Enter number of processes
4
Enter process size
3
2
4
1




Memory          15      17      23      27
P. Alloc.       P1      P2      P3      P4
Int. Frag.      12      15      19      26

Available Memory: 0
Total Internal Fragmentation: 72

--------------------------------
Process exited after 65.53 seconds with return value 0
Press any key to continue . . .
```

Program – 11

AIM: **C++ implementation of External fragmentation.**

Source Code:

```cpp
#include<iostream>

#include<algorithm>

using namespace std;

struct node{

    int memsize;

    int allocp=-1;

    int pos;

    int allocSize;

}m[200];

bool posSort(node a,node b){

    return a.pos < b.pos;

}

bool memSort(node a,node b){

    return a.memsize < b.memsize;

}

int main()

{

    int nm,np,choice, i, j, p[200];

    cout<<"Enter number of blocks\n";

    cin>>nm;

    cout<<"Enter block size\n";
```

Mridul (8521174)

```cpp
for(i=0;i<nm;i++){

    cin>>m[i].memsize;

    m[i].pos=i;

}

cout<<"Enter number of processes\n";

cin>>np;

cout<<"Enter process size\n";

for(i=0;i<np;i++){

    cin>>p[i];

}

cout<<"\n\n";

int globalFlag=0;

int pos = -1;

for(i=0;i<np;i++){

    int flag=0;

    for(j=pos+1;j<nm;j++){

        if(j==nm){

            j=0;

        }

        if(j==pos)

            break;


        if(p[i]<=m[j].memsize && m[j].allocp==-1){

            m[j].allocp=i;
```

```cpp
            m[j].allocSize=p[i];

            flag=1;

            pos = j;

            if(j==nm-1){

               j=0;

               pos = -1;

            }

            break;

         }

      }

   if(flag==0){

         cout<<"Unallocated Process P"<<i+1<<"\n";

         globalFlag=1;

      }

   }

sort(m,m+nm,posSort);

cout<<"\n";

int intFrag=0,extFrag=0;

cout<<"Memory\t\t";

for(i=0;i<nm;i++){

   cout<<m[i].memsize<<"\t";

}

cout<<"\n";

cout<<"P. Alloc.\t";
```

```cpp
for(i=0;i<nm;i++){

    if(m[i].allocp!=-1){

        cout<<"P"<<m[i].allocp+1<<"\t";

    }

    else{

        cout<<"Empty\t";

    }

}

cout<<"\n";

cout<<"Int. Frag.\t";

for(i=0;i<nm;i++){

    if(m[i].allocp!=-1){

        cout<<m[i].memsize-m[i].allocSize<<"\t";

        intFrag+=m[i].memsize-m[i].allocSize;

    }

    else{

        extFrag+=m[i].memsize;

        cout<<"Empty\t";

    }

}

cout<<"\n";

cout<<"\n";

if(globalFlag==1)

    cout<<"Total External Fragmentation: "<<extFrag<<"\n";
```

```cpp
    else

    {

        cout<<"Available Memory: "<<extFrag<<"\n";

    }

    cout<<"Total Internal Fragmentation: "<<intFrag<<"\n";

    return 0;

}
```

Output:

```
Enter number of blocks
3
Enter block size
100
300
200
Enter number of processes
3
Enter process size
150
250
350


Unallocated Process P2
Unallocated Process P3

Memory            100     300     200
P. Alloc.         Empty   P1      Empty
Int. Frag.        Empty   150     Empty

Total External Fragmentation: 300
Total Internal Fragmentation: 150
```

Mridul (8521174)