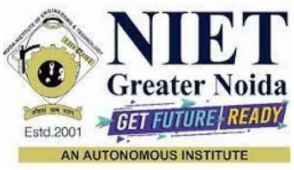


NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY
GREATER NOIDA-201306
(An Autonomous Institute)

Department of Information Technology

Session (2021 – 2022)

LAB FILE

ON

Operating Systems

(ACSE0453A)

(4th Semester)

Submitted To:

Mr. Ram Kumar Sharma
Mrs. Ayesha

Submitted By:

Name: AMAN KUMAR
Roll No: 2101330139003

Affiliated to Dr. A.P.J Abdul Kalam Technical University, Uttar Pradesh, Lucknow.

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY
GREATER NOIDA-201306
(An Autonomous Institute)

Session 2021-22

Course & Branch: B.Tech (IT) Semester : IV

Lab Name:-Operating Systems Lab Lab Code:- ACSE0453A

INDEX

S.NO	PRACTICAL CONDUCTED	DATE	PAGE NO	SIGNATURE
1	Unix and Unix Commands			
2	Implementation of System Call			
3	Introduction of C programming			
4	Implementation of FCFS CPU Scheduling Algorithm			
5	Implementation of SJF CPU Scheduling Algorithm			
6	Implementation of Priority CPU Scheduling Algorithm			
7	Implementation of RR CPU Scheduling Algorithm			
8	Implementation of Multilevel Queue PU Scheduling Algorithm			
9	Implementation of banker Algorithm			
10	Implementation of contiguous allocation memory techniques –First Fit			
11	Implementation of contiguous allocation memory techniques –Best Fit			
12	Implementation of contiguous memory allocation techniques –First Fit			
13	Implementation of contiguous memory allocation Fixed partition Technique (MFT)			
14	Implementation of contiguous memory allocation Variable partition Technique (MFT)			

Program No 1:- Unix and Unix Commands

Unix Operating System

Unix was originally developed in 1969 by a group of AT&T employees Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna at Bell Labs.

- Several people can use a Unix computer at the same time; hence Unix is called a multiuser system.
- A user can also run multiple programs at the same time; hence Unix is a multitasking environment.

There are various Unix variants available in the market.

Solaris Unix, AIX, HP Unix and BSD are a few examples. Linux is also a flavor of Unix which is freely available.

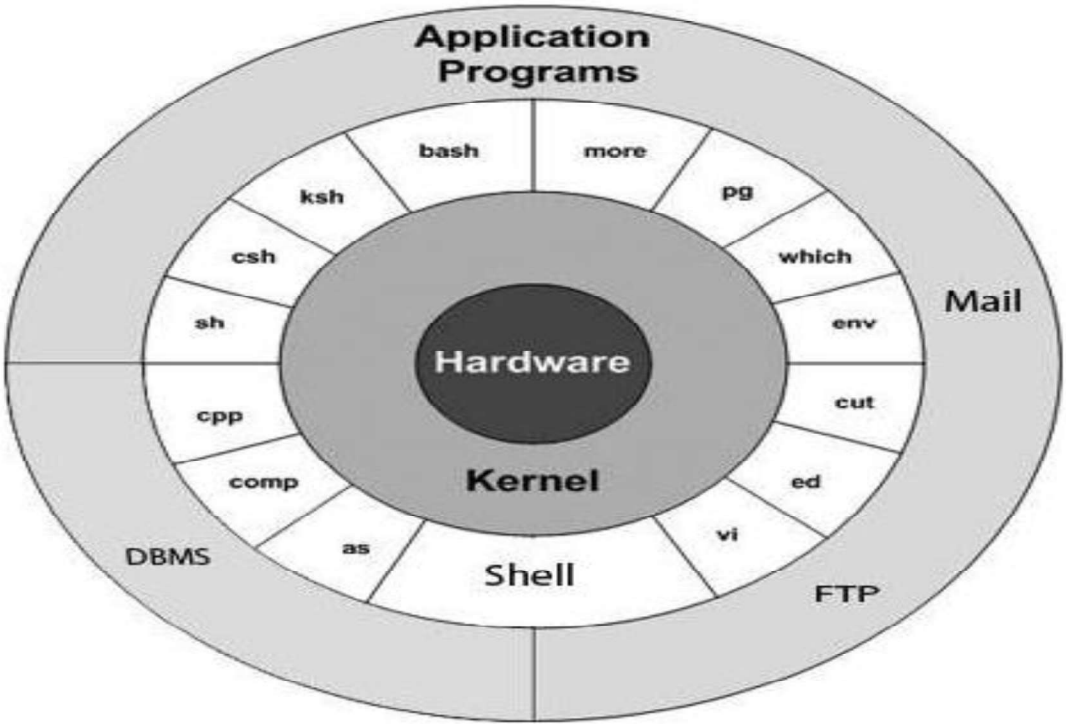
Unix Operating System

The Unix operating system is a set of programs that act as a link between the computer hardware and the user.

The computer programs that allocate the system resources and coordinate all the details of the computer's internals is called the **operating system** or the **kernel**.

Users communicate with the kernel through a program known as the **shell**. The shell is a command line interpreter; it translates commands entered by the user and converts them into a language that is understood by the kernel.

Unix Architecture :



Unix Commands

Command Name Purpose Syntax

1. **Directory Commands**

MD/MKDIR / allow user to create dir / -P -m mode directory

CD / to change the current working dir in linux and other unix-like operating system / [/D][drive:][path]

LS /List information about the file (the current directory by default)/ Is Abfs ChfsdhfgCgshglVDfsdhgfCjhfdjf FILE

DIR / to organize these files / [Drive:][Path][FileName] (/F),(/D)

RMDIR / Remove the DIRECTORY,if it is empty / OPTIONS DIRECTORY..

2. File Commands

CAT / Conceatenate file and print on the standard output / -benstuv file

TOUCH / Update the last-modified date on the given FILEs / -c -d DATE FILE FILE

CP / Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY / -fip -R -H/-L/-P SOURCE DEST

MV / Move or rename file / OPTION.. -T SOURCE DEST

RM / Remove file or directories / -dfiPRr file

MORE / View file or standard input one screenful at a time / -dlfpcsu -num+/pattern +linenum file

3.**Terminal Commands**

CLEAR / Clear the Terminal Screen / -T term

ECHO / Write arguments to the standard output / -n string

4. Help Commands

Man / An interface to the on-line reference manuals / -d -D –warning=warning -R encoding….

Help / to provide more information on another command / -d opton

5. Information commands

History / allow the user to recall, edit and return previous commands / -a

Who am I / Show who is logged on / OPTION…FILE

Tail / Print last n lines of each file to standard output / -f/-r -b number/-c number

Head / Print first 10 lines of each FILE to standard output / OPTIONS FILE

WC / Print line, word and byte counts for each FILE and a total line if more than one FILE is specified / OPTIONS FILE

Date / Display time & set time / OPTIONS+FMT TIME

CAL / calander command / cal[[month]year]

6. Permission and File Storage Commands

Chmod / Change the file mode bits of each given file according to mode / -Rcvf MODE,MODE.. FILE

Passwd / to change and sets passwords for user / -k -l -u -f -d -e -n mindays -x maxdays

Df / Display statistics about the amount of free disk space on the specified file_system or on the file system of which file is a part / df -hiKlnP -t type file/file_system

Du / see the amount of space taken by various / -aHLdclsxhmk FILE

7. Process commands

PS / Shows status of processes / -AacefbdshfsdDHdfhdb -core -Sysytem -Ofmt

Kill / send the specified signal to the specified process or process group / -s signal/-p --pid

Jobs / responsible for the installation, configuration and maintenance / jobs jobID

Program No 2 :- Implement various UNIX system call for

A. Process Management

B. Input/output System Calls

System call

The system call provides an interface to the operating system services.

Application developers often do not have direct access to the system calls, but can access them through an application programming interface (API). The functions that are included in the API invoke the actual system calls. By using the API, certain benefits can be gained:

Portability: as long a system supports an API, any program using that API can compile and run.

Ease of Use: using the API can be significantly easier than using the actual system call.

Types of System Calls : There are 5 different categories of system calls –

1. **Process control :-**This system calls perform the task of process creation, process termination, etc.
2. **File management:-**File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc.
3. **Device management:-**Device management does the job of device manipulation like reading from device buffers, writing into device buffers, etc.
4. **Information maintenance:-**It handles information and its transfer between the OS and the user program.
5. **Communication:-**These types of system calls are specially used for interprocess communications.

Program A: To Write a program using the Process system call of UNIX system call.

```
#include <stdio.h>

#include <sys/types.h>
```

```
#include <unistd.h>
int main()
{

    // make two process which run same
    // program after this instruction
    fork();

    printf("Hello world!\n");
    return 0;
}
```

Program B: To Write a program using the I/O System call of UNIX Operating system (open, read, write) etc.

2. Using system calls write line of texts in a file

```
#include<stdio.h>
#include<fcntl.h>
#include<string.h>
main()
{
char *buff,fn[10];
int fd,n,i;
printf("\nEnter file name ");
scanf("%s",fn);
printf("\nEnter text ");
scanf("%s",buff);
//fgets(buff,100,stdin);
fd=open(fn,O_CREAT|O_WRONLY);
n=write(fd,buff,strlen(buff));
close(fd);
}
```

OUTPUT:

```
Enter the file name : New
Enter the text      : hello what are you doing
```

Program No 3:- Introduction of C programming

1. INTRODUCTION OF C PROGRAMMING LANGUAGE

The C language is a structure oriented programming language, developed at Bell Laboratories in 1972 by Dennis Ritchie

C programming language features were derived from an earlier language called "B" (Basic Combined Programming Language – BCPL)

2. FEATURES OF C LANGUAGE:

C language is one of the powerful languages. Below are some of the features of C language.

Reliability

Portability

Flexibility

Interactivity

Modularity

Efficiency and Effectiveness

3. LEVEL IS C LANGUAGE BELONGING TO?

There are 3 levels of programming languages. They are,

A. Middle Level languages:

Middle level languages don't provide all the built-in functions found in high level languages, but provides all building blocks that we need to produce the result we want. Examples: C, C++

B. High Level languages:

High level languages provide almost everything that the programmer might need to do as already built into the language. Example: Java, Python

C. Low Level languages:

Low level languages provide nothing other than access to the machines basic instruction set. Example: Assembler

4. C LANGUAGE IS A STRUCTURED LANGUAGE:

Structure oriented language:

In this type of language, large programs are divided into small programs called functions

Prime focus is on functions and procedures that operate on the data Data moves freely around the systems from one function to another Program structure follows “Top Down Approach”

Examples: C, Pascal, ALGOL and Modula-2

Object oriented language:

In this type of language, programs are divided into objects

Prime focus is in the data that is being operated and not on the functions or procedures

Data is hidden and cannot be accessed by external functions

Program structure follows “Bottom UP Approach”

Examples: C++, JAVA and C# (C sharp)

Non structure oriented language:

There is no specific structure for programming this language. Examples: BASIC, COBOL, FORTRAN

5. TO WRITE A C PROGRAM:

Below are few commands and syntax used in C programming to write a simple C program. Let’s see all the sections of a simple C program line by line.

C Basic commands Explanation

#include <stdio.h> This is a preprocessor command that includes standard input output header file(stdio.h) from the C library before compiling a C program

int main() This is the main function from where execution of any C program begins.

{ This indicates the beginning of the main function.

/*_some_comments_*/ whatever is given inside the command “/* */” in any C program, won’t be considered for compilation and execution.

printf(“Hello_World! “); printf command prints the output onto the screen.

getch(); This command waits for any character input from keyboard.

return 0; This command terminates C program (main function) and returns 0.

} This indicates the end of the main function.

Program

Program 1:- Print the message Hello world

```
#include <stdio.h>
int main()
{
  /* Our first simple C basic program */
  printf("Hello World! ");
  getch();
  return 0;
}
```

OUTPUT:

```
Hello World!
```

3. STEPS TO WRITE C PROGRAMS AND GET THE OUTPUT:

Below are the steps to be followed for any C program to create and get the output. This is common to all C program and there is no exception whether its a very small C program or very large C program.

1. Create
2. Compile
3. Execute or Run
4. Get the Output

4. CREATION, COMPILATION AND EXECUTION OF A C PROGRAM:

Prerequisite:

- If you want to create, compile and execute C programs by your own, you have to install C compiler in your machine. Then, you can start to execute your own C programs in your machine.
- You can refer below link for how to install C compiler and compile and execute C programs in your machine.
- Once C compiler is installed in your machine, you can create, compile and execute C programs as shown in below link.

Program 2 . C program print integer

This c program first inputs an integer and then prints it. Input is done using scanf function and number is printed on screen using printf.

```
#include <stdio.h>
int main()
{
int a;
printf("Enter an integer\n");
scanf("%d", &a);
printf("Integer that you have entered is %d\n", a);
return 0;
}
```

Program No 3:- Add two numbers

```
#include<stdio.h>
main()
{
int a, b, c;
printf("Enter two numbers to
add\n"); scanf("%d%d",&a,&b);
c = a + b;
printf("Sum of entered numbers = %d\n",c);
return 0;
}
```

Program 4 : c printf() function

```
1  #include <stdio.h>
2  int main()
3  {
4     char ch = 'A';
5     char str[20] = "fresh2refresh.com";
6     float flt = 10.234;
7     int no = 150;
8     double dbl = 20.123456;
9     printf("Character is %c \n", ch);
10    printf("String is %s \n" , str);
11    printf("Float value is %f \n", flt);
12    printf("Integer value is %d\n" , no);
13    printf("Double value is %lf \n", dbl);
14    printf("Octal value is %o \n", no);
15    printf("Hexadecimal value is %x \n", no);
16    return 0;
```

```
17 }
```

OUTPUT:

```
Character is A
String is fresh2refresh.com
Float value is 10.234000
Integer value is 150
Double value is 20.123456
Octal value is 226
Hexadecimal value is 96
```

Program 5: for c arithmetic operators:

```
1  #include <stdio.h>
2
3  int main()
4  {
5     int a=40,b=20, add,sub,mul,div,mod;
6     add = a+b;
7     sub = a-b;
8     mul = a*b;
9     div = a/b;
10    mod = a%b;
11    printf("Addition of a, b is : %d\n", add);
12    printf("Subtraction of a, b is : %d\n", sub);
13    printf("Multiplication of a, b is : %d\n", mul);
14    printf("Division of a, b is : %d\n", div);
15    printf("Modulus of a, b is : %d\n", mod);
16 }
```

OUTPUT:

```
Addition of a, b is : 60
Subtraction of a, b is : 20
Multiplication of a, b is : 800
Division of a, b is : 2
Modulus of a, b is : 0
```

PROGRAM 6 :LOGICAL OPERATORS IN C:

```
1  #include <stdio.h>
2
3  int main()
4  {
5     int m=40,n=20;
6     int o=20,p=30;
7     if (m>n && m !=0)
```

```
 8    {
 9      printf("&& Operator : Both conditions are true\n");
10    }
11    if (o>p || p!=20)
12    {
13      printf("|| Operator : Only one condition is true\n");
14    }
15    if (!(m>n && m !=0))
16    {
17      printf("! Operator : Both conditions are true\n");
18    }
19    else
20    {
21      printf("! Operator : Both conditions are true. " \
22      "But, status is inverted as false\n");
23    }
24 }
```

OUTPUT:

```
&& Operator : Both conditions are true
|| Operator : Only one condition is true
! Operator : Both conditions are true. But, status is inverted as false
```

PROGRAM 7- FOR CONDITIONAL/TERNARY OPERATORS IN C:

```
1 #include <stdio.h>
2
3 int main()
4 {
5    int x=1, y ;
6    y = ( x ==1 ? 2 : 0 ) ;
7    printf("x value is %d\n", x);
8    printf("y value is %d", y);
9 }
```

OUTPUT:

```
x value is 1
y value is 2
```

PROGRAM 8 :- FOR IF STATEMENT IN C:

```
#include <stdio.h>
int main()
{
  int m=40,n=40;
  if (m == n)
  {
```

```
  printf("m and n are equal");
  }
}
```

OUTPUT:

```
m and n are equal
```

PROGRAM 9: FOR IF ELSE STATEMENT IN C:

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4    int m=40,n=20;
 5    if (m == n)
 6    {
 7    printf("m and n are equal");
 8    }
 9    else
10    {
11    printf("m and n are not equal");
12    }
13
14 }
```

OUTPUT:

```
m and n are not equal
```

PROGRAM 10 :- FOR NESTED IF STATEMENT IN C:

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4    int m=40,n=20;
 5    if (m>n) {
 6   printf("m is greater than n");
 7    }
 8    else if(m<n) {
 9    printf("m is less than n");
10    }
11    else {
12    printf("m is equal to n");
13    }
14 }
```

OUTPUT:

```
m is greater than n
```

PROGRAM 11:- (FOR LOOP) IN C:

In for loop control statement, loop is executed until condition becomes false.

```
include <stdio.h>

int main()
{
  int i;

  for(i=0;i<10;i++)
  {
     printf("%d ",i);
  }

}
```

OUTPUT:

```
0 1 2 3 4 5 6 7 8 9
```

PROGRAM 12:- (WHILE LOOP) IN C:

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int i=3;
6
7    while(i<10)
8    {
9      printf("%d\n",i);
10     i++;
11   }
12
13 }
```

OUTPUT:

```
3 4 5 6 7 8 9
```

PROGRAM 13:- (DO WHILE LOOP) IN C:

```
1  #include <stdio.h>
2
3  int main()
4  {
```

```
 5   int i=1;
 6
 7   do
 8   {
 9      printf("Value of i is %d\n",i);
10      i++;
11   }while(i<=4 && i>=2);
12
13 }
```

OUTPUT:

```
Value of i is 1
Value of i is 2
Value of i is 3
Value of i is 4
```

Case control statements

PROGRAM 14 :- FOR SWITCH..CASE STATEMENT IN C:

```
 1  #include <stdio.h>
 2
 3  int main ()
 4  {
 5    int value = 3;
 6    switch(value)
 7    {
 8      case 1:
 9      printf("Value is 1 \n" );
10      break;
11
12      case 2:
13      printf("Value is 2 \n" );
14      break;
15
16      case 3:
17      printf("Value is 3 \n" );
18      break;
19
20      case 4:
21      printf("Value is 4 \n" );
22      break;
23
24      default :
25      printf("Value is other than 1,2,3,4 \n" );
26    }
27    return 0;
```

```
28 }
```

OUTPUT:

```
Value is 3
```

PROGRAM 15 :- FOR BREAK STATEMENT IN C:

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4     int i;
 5
 6     for(i=0;i<10;i++)
 7     {
 8       if(i==5)
 9       {
10          printf("\nComing out of for loop when i = 5");
11          break;
12       }
13       printf("%d ",i);
14     }
15 }
```

OUTPUT:

```
0 1 2 3 4
Coming out of for loop when i = 5
```

PROGRAM 16: FOR CONTINUE STATEMENT IN C:

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4     int i;
 5     for(i=0;i<10;i++)
 6     {
 7       if(i==5 || i==6)
 8       {
 9         printf("\nSkipping %d from display using " \
10         "continue statement \n",i);
11         continue;
12       }
13       printf("%d ",i);
14     }
15 }
```

OUTPUT:

```
0 1 2 3 4
Skipping 5 from display using continue statement
Skipping 6 from display using continue statement
7 8 9
```

PROGRAM 17 :- FOR GOTO STATEMENT IN C:

```
1  #include <stdio.h>
2  int main()
3  {
4     int i;
5     for(i=0;i<10;i++)
6       {
7     if(i==5)
8     {
9       printf("\nWe are using goto statement when i = 5");
10      goto HAI;
11    }
12    printf("%d ",i);
13 }
14
15 HAI : printf("\nNow, we are inside label name \"hai\" \n");
16 }
```

OUTPUT:

```
0 1 2 3 4
We are using goto statement when i = 5
Now, we are inside label name “hai”
```

PROGRAM 18 : FOR ONE DIMENSIONAL ARRAY IN C:

```
1  #include<stdio.h>
2
3  int main()
4  {
5      int i;
6      int arr[5] = {10,20,30,40,50};
7
8         // declaring and Initializing array in C
9         //To initialize all array elements to 0, use int arr[5]={0};
10        /* Above array can be initialized as below also
11        arr[0] = 10;
12        arr[1] = 20;
13        arr[2] = 30;
14        arr[3] = 40;
15        arr[4] = 50; */
```

```
16
17      for (i=0;i<5;i++)
18      {
19          // Accessing each variable
20          printf("value of arr[%d] is %d \n", i, arr[i]);
21       }
22
23 }
```

OUTPUT:

```
value of arr[0] is 10
value of arr[1] is 20
value of arr[2] is 30
value of arr[3] is 40
value of arr[4] is 50
```

PROGRAM 19 :- FOR TWO DIMENSIONAL ARRAY IN C:

```
 1  #include<stdio.h>
 2
 3  int main()
 4  {
 5     int i,j;
 6     // declaring and Initializing array
 7     int arr[2][2] = {10,20,30,40};
 8     /* Above array can be initialized as below also
 9        arr[0][0] = 10; // Initializing array
10        arr[0][1] = 20;
11        arr[1][0] = 30;
12        arr[1][1] = 40; */
13     for (i=0;i<2;i++)
14     {
15        for (j=0;j<2;j++)
16        {
17           // Accessing variables
18           printf("value of arr[%d] [%d] : %d\n",i,j,arr[i][j]);
19        }
20     }
21 }
```

OUTPUT:

```
value of arr[0] [0] is 10
value of arr[0] [1] is 20
value of arr[1] [0] is 30
value of arr[1] [1] is 40
```

PROGRAM 20 :- FOR STRING IN C:

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

int main( )
{
  char source[ ] = " fresh2refresh" ;
  char target[ ]= " C tutorial" ;

  printf ( "\nSource string = %s", source ) ;
  printf ( "\nTarget string = %s", target ) ;

  strcat ( target, source ) ;

  printf ( "\nTarget string after strcat( ) = %s", target ) ;
}
```

OUTPUT:

```
Source string                  = fresh2refresh
Target string                  = C tutorial
Target string after strcat( ) = C tutorial fresh2refresh
```

PROGRAM 21:- FOR C FUNCTION:

```
#include<stdio.h>
// function prototype, also called function declaration
float square ( float x );
// main function, program starts from here

int main( )
{
    float m, n ;
    printf ( "\nEnter some number for finding square \n");
    scanf ( "%f", &m ) ;
    // function call
    n = square ( m ) ;
    printf ( "\nSquare of the given number %f is %f",m,n );
}

float square ( float x )   // function definition
{
    float p ;
    p = x * x ;
```

```
20     return ( p ) ;
21 }
```

OUTPUT:

```
Enter some number for finding square
2
Square of the given number 2.000000 is 4.000000
```

PROGRAM 22 :- FOR C FUNCTION (USING CALL BY VALUE):

```
 1  #include<stdio.h>
 2  // function prototype, also called function declaration
 3  void swap(int a, int b);
 4
 5  int main()
 6  {
 7     int m = 22, n = 44;
 8     // calling swap function by value
 9     printf(" values before swap  m = %d \nand n = %d", m, n);
10     swap(m, n);
11 }
12
13 void swap(int a, int b)
14 {
15     int tmp;
16     tmp = a;
17     a = b;
18     b = tmp;
19     printf(" \nvalues after swap m = %d\n and n = %d", a, b);
20 }
```

OUTPUT:

```
values before swap m = 22
and n = 44
values after swap m = 44
and n = 22
```

PROGRAM 23 :- FOR C FUNCTION (USING CALL BY REFERENCE):

```
 1  #include<stdio.h>
 2  // function prototype, also called function declaration
 3  void swap(int *a, int *b);
 4
 5  int main()
 6  {
 7     int m = 22, n = 44;
 8     //  calling swap function by reference
 9     printf("values before swap m = %d \n and n = %d",m,n);
```

```
10     swap(&m, &n);
11 }
12
13 void swap(int *a, int *b)
14 {
15     int tmp;
16     tmp = *a;
17     *a = *b;
18     *b = tmp;
19     printf("\n values after swap a = %d \nand b = %d", *a, *b);
20 }
```

OUTPUT:

```
values before swap m = 22
and n = 44
values after swap a = 44
and b = 22
```

PROGRAM 24 :- FOR POINTERS IN C:

```
#include <stdio.h>
int main()
{
  int *ptr, q;
  q = 50;
  /* address of q is assigned to ptr */
  ptr = &q;
  /* display q's value using ptr variable */
  printf("%d", *ptr);
  return 0;
}
```

S.No: 5	Exp. Name: ***Implement CPU Scheduling Algorithms***	Date: 2022-03-30

Aim:

Write a program to implement the FCFS process scheduling algorithm.

Source Code:

os.c

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

int main()
{
   char pn[10][10],t[10];
   int arr[10],bur[10],star[10],finish[10],tat[10],wt[10],i,j,n,temp;
   int totwt=0,tottat=0;
   printf("Enter the number of processes: ");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("Enter the Process Name, Arrival Time & Burst Time:");
      scanf("%s%d%d",pn[i],&arr[i],&bur[i]);

   }
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         if(arr[i]<arr[j])
         {
            temp=arr[i];
            arr[i]=arr[j];
            arr[j]=temp;
            temp=bur[i];
            bur[i]=bur[j];
            bur[j]=temp;
            strcpy(t,pn[i]);
            strcpy(pn[i],pn[j]);
            strcpy(pn[j],t);
         }
      }
   }
   for(i=0;i<n;i++)
   {
      if(i==0)
      star[i]=arr[i];
      else
      star[i]=finish[i-1];
      wt[i]=star[i]-arr[i];
      finish[i]=star[i]+bur[i];
      tat[i]=finish[i]-arr[i];

   }
```

```
   printf("Process Name\tArrival Time\tBurst Time");
   for(i=0;i<n;i++)
   {
      printf("\n%5s\t%10d\t%10d",pn[i],arr[i],bur[i]);
   }
   printf("\nPName     Arrtime     Bursttime    Start     WT   TAT   Finish");
   for(i=0;i<n;i++)
   {

      printf("\n%s\t%4d\t\t%4d\t%4d\t%4d\t%4d\t%4d",pn[i],arr[i],bur[i],star[i],wt[i],t
at[i],finish[i]);
      totwt+=wt[i];
      tottat+=tat[i];
   }
   printf("\nAverage Waiting time:%f",(float)totwt/n);
   printf("\nAverage Turn Around Time:%f",(float)tottat/n);
   return 0;
}
```


Execution Results - All test cases have succeeded!

Test Case - 1						
User Output						
Enter the number of processes: 2						
Enter the Process Name, Arrival Time & Burst Time: 1 24 27						
Enter the Process Name, Arrival Time & Burst Time: 1 26 27						
Process Name	Arrival Time	Burst Time				
1	24	27				
1	26	27				
PName	Arrtime	Bursttime	Start	WT	TAT	Finish
1	24	27	24	0	27	51
1	26	27	51	25	52	78
Average Waiting time:12.500000						
Average Turn Around Time:39.500000						

S.No: 4	Exp. Name: ***Implement CPU Scheduling Algorithms***	Date: 2022-04-05

Aim:

Write a program to implement the SJF Scheduling Algorithm.

Source Code:

os2.c

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
   int et[20],at[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10];
   int totwt=0, totta=0;
   float awt,ata;
   char pn[10][10],t[10];
   printf("Enter the number of process:");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("Enter process name, arrival time & execution time:");
      scanf("%s%d%d",pn[i],&at[i],&et[i]);
   }
   for(i=0;i<n;i++)
   for(j=0;j<n;j++)
   {
      if(et[i]<et[j])
      {
         temp=at[i];
         at[i]=at[j];
         at[j]=temp;
         temp=et[i];
         et[i]=et[j];
         et[j]=temp;
         strcpy(t,pn[i]);
         strcpy(pn[i],pn[j]);
         strcpy(pn[j],t);

      }
   }
   for(i=0;i<n;i++)
   {
      if(i==0)
      st[i]=at[i];
      else
      st[i]=ft[i-1];
      wt[i]=st[i]-at[i];
      ft[i]=st[i]+et[i];
      ta[i]=ft[i]-at[i];
      totwt+=wt[i];
      totta+=ta[i];
   }
   awt=(float)totwt/n;
```

Page No:

ID: Not updated

2020-24-IT-C2

Noida Institute of Engineering and Technology

```
   ata=(float)totta/n;
   printf("Pname\tarrivaltime\texecutiontime\twaitingtime\ttatime");
   for(i=0;i<n;i++)
   printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],wt[i],ta[i]);
   printf("\nAverage waiting time is:%f",awt);
   printf("\nAverage turnaroundtime is:%f",ata);
}
```


Execution Results - All test cases have succeeded!

Test Case - 1				
User Output				
Enter the number of process: 2				
Enter process name, arrival time & execution time: first 23 24				
Enter process name, arrival time & execution time: second 25 26				
Pname	arrivaltime	executiontime	waitingtime	tatime
first	23	24	0	24
second	25	26	22	48
Average waiting time is:11.000000				
Average turnaroundtime is:36.000000				

S.No: 3	Exp. Name: ***Implement CPU Scheduling Algorithms***	Date: 2022-04-19

Aim:

Write a program to implement the PRIORITY based cpu scheduling algorithm.

Source Code:

os3.c

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
   int et[20],at[10],n,i,j,temp,p[10],st[10],ft[10],wt[10],ta[10];
   int totwt=0, totta=0;
   float awt,ata;
   char pn[10][10],t[10];
   printf("Enter the number of process:");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("Enter process name,arrivaltime,execution time & priority:");
      scanf("%s%d%d%d",pn[i],&at[i],&et[i],&p[i]);
   }
   for(i=0;i<n;i++)

   for(j=0;j<n;j++)
   {
      if(p[i]<p[j])
      {
         temp=p[i];
         p[i]=p[j];
         p[j]=temp;
         temp=at[i];
         at[i]=at[j];
         at[j]=temp;
         temp=et[i];
         et[i]=et[j];
         et[j]=temp;
         strcpy(t,pn[i]);
         strcpy(pn[i],pn[j]);
         strcpy(pn[j],t);
      }
   }
   for(i=0;i<n;i++)
   {
      if(i==0)
      {
         st[i]=at[i];
         wt[i]=st[i]-at[i];
         ft[i]=st[i]+et[i];
         ta[i]=ft[i]-at[i];
```

```
      }
      else
      {
         st[i]=ft[i-1];
         wt[i]=st[i]-at[i];
         ft[i]=st[i]+et[i];
         ta[i]=ft[i]-at[i];

      }
      totwt+=wt[i];
      totta+=ta[i];

   }
   awt=(float)totwt/n;
   ata=(float)totta/n;
   printf("Pname\tarrivaltime\texecutiontime\tpriority\twaitingtime\ttatime");
   for(i=0;i<n;i++)
   printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],p[i],wt[i],ta[i]);
   printf("\nAverage waiting time is:%f",awt);
   printf("\nAverage turnaroundtime is:%f",ata);
}
```


Execution Results - All test cases have succeeded!

Test Case - 1					
User Output					
Enter the number of process: 2					
Enter process name,arrivaltime,execution time & priority: first 4 6 7					
Enter process name,arrivaltime,execution time & priority: second 5 7 8					
Pname	arrivaltime	executiontime	priority	waitingtime	tatime
first	4	6	7	0	6
second	5	7	8	5	12
Average waiting time is:2.500000					
Average turnaroundtime is:9.000000					

S.No: 2	Exp. Name: ***Implement CPU Scheduling Algorithms***	Date: 2022-04-19

Aim:

Implementation of the Round Robin cpu scheduling algorithm
(https://gecgudlavalleru.codetantra.com/secure/labs-q.jsp?
sNo=4&qId=5bec179564bac110545ba035&bd=AY3RFZHVEQg%3D%3D&lid=5db6d168a183970b79e5cd34&labbd=AMzM2X2N0X2No&expTitle=Implementation%20of%20the%20Round%

Source Code:

os4.c

```
#include<stdio.h>
#include<conio.h>
#include<conio.h>
void main()
{
   int i, NOP, sum=0,count=0,y,quant,wt=0,tat=0,at[10],bt[10],temp[10];
   float avg_wt,avg_tat;
   printf("Enter Total Number of Processes: ");
   scanf("%d",&NOP);
   y=NOP;
   for(i=0;i<NOP;i++)
   {
      printf("Enter Details of Process[%d]: Arrival Time:\t",i+1);
      scanf("%d",&at[i]);
      printf("Burst Time:\t");
      scanf("%d",&bt[i]);
      temp[i]=bt[i];
   }
   printf("Enter Time Quantum:\t");
   scanf("%d",&quant);
   printf("Process ID\t\tBurst Time\t Turnaround Time\t Waiting Time");
   for(i=0;i<NOP;i++)
   {
      if(temp[i]<=quant&&temp[i]>0)
      {
         sum=sum+temp[i];
         temp[i]=0;
         count=1;
      }
      else if(temp[i]>0)
      {
         temp[i]=temp[i]-quant;
         sum=sum+quant;
      }
      if(temp[i]==0&&count==1)
      {
         //i--;
         printf("\nProcess[%d]\t\t%d\t\t\ %d\t\t\t %d",i+1,bt[i],sum-at[i],sum-at[i]-bt
[i]);
         wt=wt+sum-at[i]-bt[i];
         tat=tat+sum-at[i];
         count=0;
         i--;
      }
      if(i==NOP-1)
      {
         i=0;
      }
      else if(at[i+1]<=sum)
      {
         i++;
      }
      else
      {
         i=0;
      }
   }
   avg_wt=wt*1.0/NOP;
   avg_tat=tat*1.0/NOP;
   printf("\nAverage Waiting Time:\t%f",avg_wt);
   printf("\nAvg Turnaround Time:\t%f\n",avg_tat);
   }
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter Total Number of Processes: 3
Enter Details of Process[1]: Arrival Time: 0

Test Case - 1

```
Burst Time:      3
Enter Details of Process[2]: Arrival Time:       0
Burst Time:      2
Enter Details of Process[3]: Arrival Time:       1
Burst Time:      3
Enter Time Quantum:      5
```

Process ID	Burst Time	Turnaround Time	Waiting Time
Process[1]	3	3	0
Process[2]	2	5	3
Process[3]	3	7	4

```
Average Waiting Time:    2.333333
Avg Turnaround Time:     5.000000
```

S.No: 1	Exp. Name: ***Implement CPU Scheduling Algorithms***	Date: 2022-04-19

Aim:

Write a program to implement the Multi Level Queue Scheduling.

Source Code:

os5.c

```
#include<stdio.h>
int main()
{
   int p[20],bt[20],su[20],wt[20],tat[20],i,k,n,temp;
   float wtavg,tatavg;
   printf("Enter the number of processes:");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      p[i]=i;
      printf("Enter the Burst Time of Process %d:",i);
      scanf("%d",&bt[i]);
      printf("System/User Process (0/1) ?");
      scanf("%d",&su[i]);
   }
   for(i=1;i<n;i++)
   for(k=i+1;k<n;k++)
   if(su[i]>su[k])
   {
      temp=p[i];
      p[i]=p[k];
      p[k]=temp;
      temp=bt[i];
      bt[i]=bt[k];
      bt[k]=temp;
      temp=su[i];
      su[i]=su[k];
      su[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("PROCESS\t\t SYSTEM/USER PROCESS \tBURST TIME\tWAITING TIME\tTURNAROUND TIM
E");
   for(i=0;i<n;i++)
   printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],su[i],bt[i],wt[i],tat[i]);
   printf("\nAverage Waiting Time is --- %f" ,wtavg/n);
   printf("\nAverage Turnaround Time is --- %f",tatavg/n);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1				
User Output				
Enter the number of processes: 2				
Enter the Burst Time of Process 0: 45				
System/User Process (0/1) ? 0				
Enter the Burst Time of Process 1: 67				
System/User Process (0/1) ? 1				
PROCESS	SYSTEM/USER PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
0	0	45	0	45
1	1	67	45	112
Average Waiting Time is --- 22.500000				
Average Turnaround Time is --- 78.500000				

Page No:

ID: Not updated

2020-24-IT-C2

Noida Institute of Engineering and Technology

S.No: 6	Exp. Name: ***Write the code to implement Banker's Algorithm***	Date: 2022-04-19

Aim:

Write the C program to implement Banker's Algorithm

Source Code:

bankersAlgorithm.c

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int n,r,i,j,k,p,u=0,s=0,m;
   int block[10],run[10],active[10],newreq[10];
   int max[10][10],resalloc[10][10],resreq[10][10];
   int totalloc[10],totext[10],simalloc[10];
   printf("Enter the no of processes: ");
   scanf("%d",&n);
   printf("Enter the no of resource classes: ");
   scanf("%d",&r);
   printf("Enter the total existed resource in each class: ");
   for(k=1;k<=r;k++)
   scanf("%d",&totext[k]);
   printf("Enter the allocated resources: ");
   for(i=1;i<=n;i++)
   for(k=1;k<=r;k++)
   scanf("%d",&resalloc);
   printf("Enter the process making the new request: ");
   scanf("%d",&p);
   printf("Enter the requested resource: ");
   for(k=1;k<=r;k++)
   scanf("%d",&newreq[k]);
   printf("Enter the process which are n blocked or running\n");
   for(i=1;i<=n;i++)
   {
      if(i!=p)
      {
         printf("process %d: \n",i+1);
         scanf("%d%d",&block[i],&run[i]);
      }
   }
   block[p]=0;
   run[p]=0;
   for(k=1;k<=r;k++)
   {
      j=0;
      for(i=1;i<=n;i++)
      {
         totalloc[k]=j+resalloc[i][k];
         j=totalloc[k];
      }
   }
   for(i=1;i<=n;i++)
   {
```

```
   if(block[i]==1||run[i]==1)
   active[i]=1;
   else
   active[i]=0;
}
for(k=1;k<=r;k++)
{
   resalloc[p][k]+=newreq[k];
   totalloc[k]+=newreq[k];
}
for(k=1;k<=r;k++)
{
   if(totext[k]-totalloc[k]<0)
   {
      u = 1;
      break;
   }
}
if(u==0)
{
   for(k=1;k<=r;k++)
   simalloc[k]=totalloc[k];
   for(s=1;s<=n;s++)
   for(i=1;i<=n;i++)
   {
      if(active[i]==1)
      {
         j=0;
         for(k=1;k<=r;k++)
         {
            if((totext[k]-simalloc[k])<(max[i][k]-resalloc[i][k]))
            {
               j=1;
               break;
            }
         }
      }
      if(j==0)
      {
         active[i];
         for(k=1;k<=r;k++)
         simalloc[k]=resalloc[i][k];

      }
   }
   m=0;
   for(k=1;k<=r;k++)
   resreq[p][k]=newreq[k];
   printf("Deadlock willn't occur\n");
}
else
{
   for(k=1;k<=r;k++)
   {
      resalloc[p][k]=newreq[k];
      totalloc[k]=newreq[k];
```

```
      }
      printf("Deadlock will occur\n");
   }
   return 0;
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the no of processes: 2
Enter the no of resource classes: 2
Enter the total existed resource in each class: 2 4 3 7
Enter the allocated resources: 5 9
Enter the process making the new request: 2 6
Enter the requested resource: 5 3
Enter the process which are n blocked or running 2 6
process 2: 2 6
Deadlock will occur

Test Case - 2
User Output
Enter the no of processes: 1
Enter the no of resource classes: 1
Enter the total existed resource in each class: 1
Enter the allocated resources: 1
Enter the process making the new request: 1
Enter the requested resource: 1
Enter the process which are n blocked or running
Deadlock willn't occur

S.No: 7	Exp. Name: ***Write the code to implement the Contiguous allocation technique: - First-Fit***	Date: 2022-04-19

Aim:

Write a C program to implement the Contiguous allocation technique: - First-Fit

Source Code:

contiguousAllocationTechnique.c

```
#include<stdio.h>
#include<conio.h>
#define max 25

int main()
{
   int frag[max],b[max],f[max],i,j,nb,nf,temp;
   static int bf[max],ff[max];

   printf("Enter the number of blocks: ");
   scanf("%d",&nb);
   printf("Enter the number of files: ");
   scanf("%d",&nf);
   printf("Enter the size of the blocks\n");
   for(i=1;i<=nb;i++)
   {
      printf("Block %d: ",i);
      scanf("%d",&b[i]);
   }
   printf("Enter the size of the files\n");
   for(i=1;i<=nf;i++)
   {
      printf("File %d: ",i);
      scanf("%d",&f[i]);
   }
   for (i=1;i<=nf;i++)
   {
      for(j=1;j<=nb;j++)
      {
         if(bf[j]!=1)
         {
            temp=b[j]-f[i];
            if(temp>=0)
            {
               ff[i]=j;
               break;
            }
         }
      }
      frag[i]=temp;
      bf[ff[i]]=1;
   }
   printf("File_no\tFile_size\tBlock_no\tBlock_size\tFragement\n");
   for(i=1;i<=nf;i++)
       printf("%d\t%d\t%d\t%d\t%d\n",i,f[i],ff[i],b[ff[i]],frag[i]);
```

```
        return 0;
}
```


Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the number of blocks:  3
Enter the number of files:  2
Enter the size of the blocks 5
Block 1:  5
Block 2:  1
Block 3:  4
Enter the size of the files 2
File 1:  2
File 2:  4
File_no File_size       Block_no        Block_size      Fragement
1       2       1       5       3
2       4       3       4       0
```

Test Case - 2

User Output

```
Enter the number of blocks:  4
Enter the number of files:  6
Enter the size of the blocks 2
Block 1:  2
Block 2:  6
Block 3:  1
Block 4:  8
Enter the size of the files 6
File 1:  6
File 2:  8
File 3:  1
File 4:  3
File 5:  5
File 6:  9
File_no File_size       Block_no        Block_size      Fragement
1       6       2       6       0
2       8       4       8       0
3       1       1       2       1
4       3       0       144     -2
5       5       0       144     -4
6       9       0       144     -8
```

S.No: 8	Exp. Name: ***Write a program to Implementation of Contiguous allocation technique: - Best-Fit***	Date: 2022-04-21

Aim:

Write a program to Implementation of Contiguous allocation technique: - Best-Fit

Source Code:

bestFit.c

```
#include<stdio.h>
#include<conio.h>
#define max 25
int main()
{
   int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
   static int bf[max],ff[max];
   printf("Memory Management Scheme for contigus memeory allocation - Best Fit\n");
   printf("Enter the number of blocks:");
   scanf("%d",&nb);
   printf("Enter the number of files:");
   scanf("%d",&nf);
   printf("Enter the size of the blocks:-\n");
   for(i=1;i<=nb;i++)
   {
      printf("Block %d:",i);
      scanf("%d",&b[i]);
   }
   printf("Enter the size of the files :-\n");
   for(i=1;i<=nf;i++)
   {
      printf("File %d:",i);
      scanf("%d",&f[i]);
   }
   for(i=1;i<=nf;i++)
   {
      for(j=1;j<=nb;j++)
      {
         if(bf[j]!=1)
         {
            temp=b[j]-f[i];
            if(temp>=0)
            if(lowest>temp)
            {
               ff[i]=j;

               lowest=temp;
            }
         }
      }
      frag[i]=lowest;
      bf[ff[i]]=1;
      lowest=10000;
   }
   printf("File No\tFile Size \tBlock No\tBlock Size\tFragment");
   for(i=1;i<=nf&& ff[i]!=0;i++)
   printf("%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
   return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1								
User Output								
Memory Management Scheme for contigus memeory allocation - Best Fit 3								
Enter the number of blocks: 3								
Enter the number of files: 2								
Enter the size of the blocks:- 5								
Block 1: 5								
Block 2: 1								
Block 3: 4								
Enter the size of the files :- 3								
File 1: 3								
File 2: 4								
File No File Size	Block No	Block Size	Fragment1	3	3	4	12	4

S.No: 9	Exp. Name: ***Write a program to Implementation of Contiguous allocation technique :- Worst-Fit***	Date: 2022-04-22

Aim:

Write a program to Implementation of Contiguous allocation technique :- Worst-Fit

Source Code:

worsrFitAlgorithm.c

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
   int frag[max],b[7],f[max],i,j,nb,nf,temp,highest=0;
   static int bf[max],ff[max];
   printf("Enter the number of blocks: ");
   scanf("%d",&nb);
   printf("Enter the number of files: ");
   scanf("%d",&nf);
   printf("Enter the size of the blocks\n");
   for(i=1;i<=nb;i++)
   {
      printf("Block %d: ",i);
      scanf("%d",&b[i]);
   }
   printf("Enter the size of the files\n");
   for(i=1;i<=nf;i++)
   {
      printf("File %d: ",i);
      scanf("%d",&f[i]);
   }
   for(i=1;i<=nf;i++)
   {
      for(j=1;j<=nb;j++)
      {
         if(bf[j]!=1)
         {
            temp=b[j]-f[i];
            if(temp>=0)
            if(highest<temp)
            {
               ff[i]=j;
               highest=temp;
            }
         }
      }
      frag[i]=highest;
      bf[ff[i]]=1;
      highest=0;
   }
   printf("File_no\tFile_size\tBlock_no\tBlock_size\tFragement\n");
   for(i=1;i<=nf;i++)
   printf("%d\t%d\t%d\t%d\t%d\n",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the number of blocks:  4
Enter the number of files:  3
Enter the size of the blocks 5
Block 1:  5
Block 2:  4
Block 3:  3
Block 4:  5
Enter the size of the files 2
File 1:  2
File 2:  9
File 3:  4
File_no File_size       Block_no        Block_size      Fragement
1       2       1       5       3
2       9       0       0       0
3       4       4       5       1
```

Test Case - 2

User Output

```
Enter the number of blocks:  5
Enter the number of files:  7
Enter the size of the blocks 2
Block 1:  2
Block 2:  6
Block 3:  4
Block 4:  8
Block 5:  12
Enter the size of the files 36
File 1:  36
File 2:  14
File 3:  25
File 4:  4
File 5:  36
File 6:  12
File 7:  24
File_no File_size       Block_no        Block_size      Fragement
1       36      0       0       0
2       14      0       0       0
3       25      0       0       0
4       4       5       12      8
5       36      0       0       0
6       12      0       0       0
7       24      0       0       0
```

S.No: 10	Exp. Name: ***Write a program to Implementation of contiguous memory fixed partition technique(MFT)***	Date: 2022-04-22

Aim:

Write a program to Implementation of contiguous memory fixed partition technique(MFT)

Source Code:

fixedPartitionTechnique.c

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int m,p,s,p1;
   int m1[4],i,f,f1=0,f2=0,fra1,fra2,s1,pos;
   printf("Enter the memory size:");
   scanf("%d",&m);
   printf("Enter the no of partitions:");
   scanf("%d",&p);
   s=m/p;
   printf("Each partn size is:%d",s);
   printf("Enter the no of processes:");
   scanf("%d",&p1);
   pos=m;
   for(i=0;i<p1;i++)
   {
      printf("Enter the memory req for process%d:",i+1);
      scanf("%d",&m1[i]);
      if(m1[i]<=s)
      {
         printf("Process is allocated in partition%d\n",i+1);
         fra1=s-m1[i];
         printf("Internal fragmentation for process is:%d\n",fra1);
         f1=f1+fra1;
         pos=pos-s;
      }
      else
      {
         printf("Process not allocated in partition%d\n",i+1);
         s1=m1[i];
         while(s1>s)
         {
            s1=s1-s;
            pos=pos-s;
         }
         pos=pos-s;
         fra2=s;
         f2=f2+fra2;
         printf("External fragmentation for partition is:%d",fra2);
      }
   }
   printf("Process\tmemory\tallocatedmemory");
   for(i=0;i<p1;i++)
   printf("\n%5d\t%5d\t%5d",i+1,s,m1[i]);
   f=f1+f2;
```

```
   printf("\nThe tot no of fragmentation is:%d",f);
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the memory size: 500
Enter the no of partitions: 4
Each partn size is:125Enter the no of processes: 4
Enter the memory req for process1: 100
Process is allocated in partition1 200
Internal fragmentation for process is:25 200
Enter the memory req for process2: 200
Process not allocated in partition2 100
External fragmentation for partition is:125Enter the memory req for process3: 100
Process is allocated in partition3 50
Internal fragmentation for process is:25 50
Enter the memory req for process4: 50
Process is allocated in partition4
Internal fragmentation for process is:75
Process memory allocatedmemory
1 125 100
2 125 200
3 125 100
4 125 50
The tot no of fragmentation is:250

S.No: 11	Exp. Name: ***Write a program to Implementation of contiguous memory Variable partition technique (MVT)***	Date: 2022-04-22

Aim:

Write a program to Implementation of contiguous memory Variable partition technique (MVT)

Source Code:

VariablePartition.c

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int m=0,m1=0,m2=0,p,count=0,i;
   printf("enter the memory capacity:");
   scanf("%d",&m);
   printf("enter the no of processes:");
   scanf("%d",&p);
   for(i=0;i<p;i++)
   {
      printf("enter memory req for process%d:",i+1);
      scanf("%d",&m1);
      count=count+m1;
      if(m1<=m)
      {
         if(count==m)
         {
            printf("there is no further memory remaining:\n");
         }
         else
         {
            printf("the memory allocated for process%d is: %d ",i+1,m);
            m2=m-m1;
            printf("\nremaining memory is: %d\n",m2);
            m=m2;
         }
      }
      else
      {
         printf("emory is not allocated for process%d",i+1);
      }
      printf("external fragmentation for this process is:%d\n",m2);
   }
   return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
enter the memory capacity: 500

Test Case - 1
enter the no of processes: 2
enter memory req for process1: 250
the memory allocated for process1 is: 500 50
remaining memory is: 250 50
external fragmentation for this process is:250 50
enter memory req for process2: 50
the memory allocated for process2 is: 250
remaining memory is: 200
external fragmentation for this process is:200

Test Case - 2
User Output
enter the memory capacity: 250
enter the no of processes: 2
enter memory req for process1: 250
there is no further memory remaining: 120
external fragmentation for this process is:0 120
enter memory req for process2: 120
the memory allocated for process2 is: 250
remaining memory is: 130
external fragmentation for this process is:130