ADITYA COLLEGE OF ENGINEERING & TECHNOLOGY

ADB ROAD, SURAMPALEM, E.G. DIST.

DEPARTMENT OF

Name: PIN No. | | | | | | | | | | |

Certified that this is the bonafide record of practical work done by

Mr. / Ms. .....

a students of ..... with PIN No.....

in the ..... Laboratory during the year .....

No of Practicals Conducted | | No. of Practicals Attended | |

Signature - Faculty Incharge

Signature - Head of the Department

Submitted for the Practical examination held on.....

EXAMINER - 1

EXAMINER - 2

INDEX

Sl.No	Date	Name of the Experiment	Page No.

Experiment – 1
Study of Unix/Linux general purpose utility commands

a) ls Command:
The ls command is used to display a list of content of a directory.
Syntax: ls [Options]
b) clear Command:
Linux clear command is used to clear the terminal screen.
Syntax: clear
c) mkdir Command:
The mkdir command is used to create a new directory under any directory.
Syntax: mkdir <directory name>
d) cd Command:
The cd command is used to change the current directory.
Syntax: cd <directory name>
e) rmdir Command:
The rmdir command is used to delete a directory.
Syntax: rmdir <directory name>
f) touch Command:
The touch command is used to create empty files. We can create multiple empty files by executing it once.
Syntax: touch <file name>

touch <file1> <file2> ....
$ touch file2
g) cat Command:
The cat command is a multi-purpose utility in the Linux system. It can be used to create a file, display content of the file, copy the content of one file to another file, and more.
Syntax: cat [OPTION]… [FILE]..
To create a file, execute it as follows:

cat > <file name>

// Enter file content
Press "CTRL+ D" keys to save the file. To display the content of the file, execute it as follows:

cat <file name>
h) mv Command:
The mv command is used to move a file or a directory form one location to another location.
Syntax: mv <file name> <directory path>
OUTPUT:

Experiment - 2

Study of Bash shell, Bourne shell and C shell in Unix/Linux operating system.

SHELL is a program which provides the interface between the user and an operating system.
Types of Shells in Linux
Bourne shell
Bash shell
C shell
Korn shell

Bourne shell:

The Bourne shell, called "sh," is one of the original shells, developed for Unix computers by Stephen Bourne at AT&T's Bell Labs in 1977.
It offers features such as input and output redirection, shell scripting with string and integer variables, and condition testing and looping.

Bash shell:

The "Bourne-again Shell," based on sh -- has become the new default standard. One attractive feature of bash is its ability to run sh shell scripts unchanged. Shell scripts are complex sets of commands that automate programming and maintenance chores; being able to reuse these scripts saves programmers time.

C shell:

Developers have written large parts of the Linux operating system in the C and C++ languages. Using C syntax as a model, Bill Joy at Berkeley University developed the "C-shell," csh, in 1978.

Ken Greer, working at Carnegie-Mellon University, took csh concepts a step forward with a new shell, tcsh, which Linux systems now offer. Tcsh fixed problems in csh and added command completion, in which the shell makes educated "guesses" as you type, based on your system's directory structure and files.

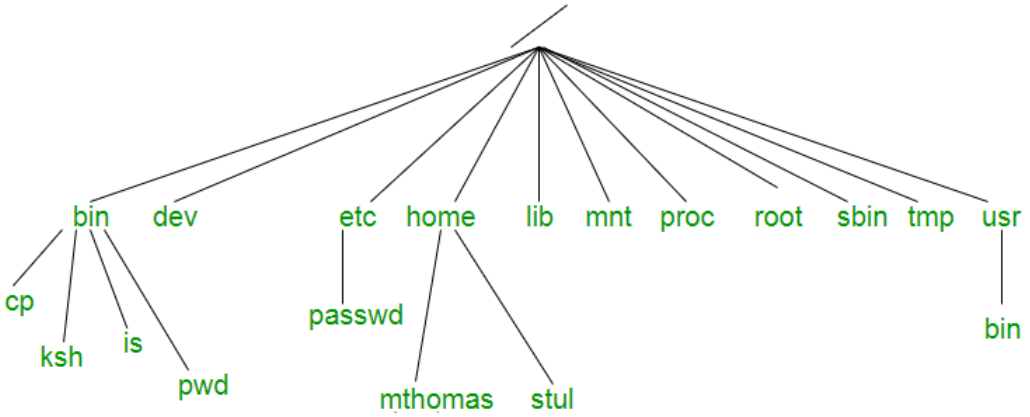
Korn shell:

David Korn developed the Korn shell, or ksh, about the time tcsh was introduced. Ksh is compatible with sh and bash. Ksh improves on the Bourne shell by adding floating-point arithmetic, job control, and command aliasing and command completion

OUTPUT:

Experiment - 3

Study of UNIX/LINUX File System(tree structure).



The UNIX file system called as hierarchical file system. The UNIX system has no limit to the number of files and directories you can create in a directory that you own.

/ (root):- This is the root directory of the file system, the main directory of the entire file system, and the root directory for the super user.

/bin:- bin stands for binary. This directory contains executable files for most of the Unix commands.

/dev: - This contains the special device files that includes terminals, printers and storage devices.

/etc: - This contains system administration and configuration databases.

/home:-This contains the home directories and files of all users. If your log name is Deepu, your default home directory is /home/Deepu.

/lib: - This directory contains all the library functions provided by UNIX for programmers.

/mnt: - Contains entries for removable media such as CD-ROMs.

/sbin: - This contains programs used in booting the system and in system recovery.

/tmp: - This contains all temporary files used by the UNIX system.

/usr: - This contains other accessible directories such as /usr/lib and /usr/bin

/var: - This contains the directories of all files that vary among systems. These include files that log system activity, accounting files, mail files, application packages, backup files for editors and many other types of files that vary from system to system.

OUTPUT:

Experiment – 4

C program to emulate the UNIX ls –l command

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
int main()
{
int pid;            //process id
pid = fork();     //create another process
if ( pid < 0 )
{                              //fail
printf("\nFork failed\n");
exit (-1);
}
else if ( pid == 0 )
{                   //child
execlp ("/bin/ls", "ls", "-l", NULL );  //execute ls
}
else
{                           //parent
wait (NULL);              //wait for child
printf("\nchild complete\n");
exit (0);
}
}
```

OUTPUT:

Experiment - 5

C program that illustrates how to execute two commands concurrently with a command pipe. Ex: - ls –l | sort

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
int main()
{
int pfds[2];
char buf[30];
if(pipe(pfds)==-1)
{
perror("pipe failed");
exit(1);
}
if(!fork())
{
close(1);
dup(pfds[1];
system (“ls –l”);
}
else
{
printf("parent reading from pipe \n");
while(read(pfds[0],buf,80))
printf("%s \n" ,buf);
}
}
```

OUTPUT:

Experiment - 6

Write a Shell program to check whether given number is prime or not.

Program:

```
echo "Prime number program....."
read number
i=2
flag=0
while [ $i -le `expr $number / 2` ]
do
if [ `expr $number % $i` -eq 0 ]
then
flag=1
fi

i=`expr $i + 1`
done
if [ $flag -eq 1 ]
then
echo "The number is Not Prime"
else
echo "The number is Prime"
fi
```

OUTPUT:

Experiment - 7

Write a shell script which will display Fibonacci series up to the given range.

Program:

```
echo "How many number of terms to be generated ?"
 read n
 x=0
 y=1
 i=2
 echo "Fibonacci Series up to $n terms :"
 echo "$x"
 echo "$y"
 while [ $i -lt $n ]
 do
    i=`expr $i + 1 `
    z=`expr $x + $y `
    echo "$z"
    x=$y
    y=$z
 done
```

OUTPUT:

Experiment - 8

Write a shell script to check whether the given number is Armstrong or not.

Program:

```
echo "Enter the number"
read n
function ams
{
t=$n
s=0
b=0
c=10
while [ $n -gt $b ]
do
r=$((n % c))
i=$((r * r * r))
s=$((s + i))
n=$((n / c))
done
echo $s
if [ $s == $t ]
then
echo "Amstrong number"
else
echo "Not an Armstrong number"
fi
}
result=`ams $n`
echo "$result"
```

OUTPUT:

Expeiment - 9

Write a shell script to accept student number, name, marks in 5 subjects. Find total, average and grade using the following rules:

Avg>=80 then grade A
Avg<80&&Avg>=70 then grade B
Avg<70&&Avg>=60 then grade C
Avg<60&&Avg>=50 then grade D
Avg<50&&Avg>=40 then grade E

Program:

```
clear
echo ----------------------------------
echo '\tStudent Mark List'
echo ----------------------------------
echo Enter the Student name
read name
echo Enter the Register number
read rno
echo Enter the Mark1
read m1
echo Enter the Mark2
read m2
echo Enter the Mark3
read m3
echo Enter the Mark4
read m4
echo Enter the Mark5
read m5
tot=$(expr $m1 + $m2 + $m3 + $m4 + $m5)
avg=$(expr $tot / 5)
echo ----------------------------------
echo '\tStudent Mark List'
echo ----------------------------------
echo "Student Name    : $name"
echo "Register Number : $rno"
echo "Mark1           : $m1"
echo "Mark2           : $m2"
echo "Mark3           : $m3"
echo "Mark4           : $m4"
echo "Mark5           : $m5"
echo "Total          : $tot"
echo "Average         : $avg"
```

```
if [ $m1 -ge 35 ] && [ $m2 -ge 35 ] && [ $m3 -ge 35 ] && [ $m4 -ge 35 ] && [ $m5 -ge 35
]
then
 echo "Result         : Pass"

 if [ $avg -ge 90 ]
 then
  echo "Grade          : S"
 elif [ $avg -ge 80 ]
 then
        echo "Grade          : A"
 elif [ $avg -ge 70 ]
 then
        echo "Grade          : B"
 elif [ $avg -ge 60 ]
 then
        echo "Grade          : C"
 elif [ $avg -ge 50 ]
 then
        echo "Grade          : D"
 elif [ $avg -ge 35 ]
 then
        echo "Grade          : E"
 fi
else
 echo "Result         : Fail"
fi
echo ----------------------------------
```

OUTPUT:

Experiment – 10

Write a shell script to find minimum and maximum elements in the given list of elements.

Program:

```
read -a integers

biggest=${integers[0]}
smallest=${integers[0]}

for i in ${integers[@]}
do
  if [[ $i -gt $biggest ]]
  then
    biggest="$i"
  fi

  if [[ $i -lt $smallest ]]
  then
    smallest="$i"
  fi
done
echo "The largerst number is $biggest"
echo "The smallest number is $smallest"
```

OUTPUT:

Experiment – 11

Write a shell program to check whether the given string is palindrome or not.

Program:

```
clear
echo "Enter the string:" read str
echo
len=`echo $str | wc -c`
len=`expr $len - 1`
i=1
j=`expr $len / 2`
while test $i -le $j do
k=`echo $str | cut -c $i` l=`echo $str | cut -c $len`
 if test $k != $l
then
echo "String is not palindrome"
exit
fi

i=`expr $i + 1`
len=`expr $len - 1`
done
echo "String is palindrome"
```

OUTPUT:

Experiment – 12

Write a shell script to compute no. of characters and words in each line of given file

Program:

```
echo Enter the filename
read file
c=`cat $file | wc -c`
w=`cat $file | wc -w`
l=`grep -c "." $file`
echo Number of characters in $file is $c
echo Number of words in $file is $w
echo Number of lines in $file is $l
```

OUTPUT:

Experiment - 13
Write a shell script to check whether the given input is a number or a string

Program:

```
read -p "Input : " inp
while [[ "$inp" =~ [^a-zA-Z0-9] || -z "$inp" ]]
do
  echo "The input contains special characters."
  echo "Input only alphanumeric characters."
  read -p "Input : " inp
done
echo "Successful Input"
```

OUTPUT:

Experiment – 14

Simulate the Following CPU Scheduling Algorithms

14.a FCFS Scheduling :

```
#include<stdio.h>
#include<conio.h>
int main() {
   int bt[20], wt[20], tat[20], i, n;
   float wtavg, tatavg;

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

   printf("\t PROCESS \t BURST 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);

   return 0;
}
```

OUTPUT:

14.b SJF Scheduling:

```
#include <stdio.h>
#include <conio.h>
int main() {
   int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
   float wtavg, tatavg;
   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]);
   }

   // Sorting processes based on burst time using bubble sort
   for (i = 0; i < n; i++) {
      for (k = i + 1; k < n; k++) {
         if (bt[i] > bt[k]) {
            // Swap burst times
            temp = bt[i];
            bt[i] = bt[k];
            bt[k] = temp;
            // Swap process IDs
            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 += wt[i];
      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);

    return 0;
}
```

OUTPUT:

14.c Priority Scheduling:

```
#include <stdio.h>

int main() {
    int p[20], bt[20], pri[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 & Priority of Process %d: ", i);
        scanf("%d %d", &bt[i], &pri[i]);
    }

    // Sorting processes based on priority using bubble sort
    for (i = 0; i < n; i++) {
        for (k = i + 1; k < n; k++) {
            if (pri[i] > pri[k]) {
                // Swap process IDs
                temp = p[i];
                p[i] = p[k];
                p[k] = temp;

                // Swap burst times
                temp = bt[i];
                bt[i] = bt[k];
                bt[k] = temp;

                // Swap priorities
                temp = pri[i];
                pri[i] = pri[k];
                pri[k] = temp;
            }
        }
    }

    wtavg = wt[0] = 0;
    tatavg = tat[0] = bt[0];

    for (i = 1; i < n; i++) {
        wt[i] = wt[i - 1] + bt[i - 1];
```

```
        tat[i] = tat[i - 1] + bt[i];
        wtavg = wtavg + wt[i];
        tatavg = tatavg + tat[i];
    }

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

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

    return 0;
}
```

OUTPUT:

14.d Round Robin Scheduling:

```
#include <stdio.h>
int main() {
   int i, j, n, bu[10], wa[10], tat[10], t, ct[10], max;
   float awt = 0, att = 0, temp = 0;

   printf("Enter the number 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 (i = 0; i < (max / t) + 1; i++) {
      for (j = 0; j < n; j++) {
         if (bu[j] != 0) {
            if (bu[j] <= t) {
               tat[j] = temp + bu[j];
               temp += bu[j];
               bu[j] = 0;
            } else {
               bu[j] -= t;
               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]);

    return 0;
}
```

OUTPUT:

Experiment - 15

Simulate The Following

15.a. Multiprogramming with A Fixed Number Of Tasks (MFT)

```
#include <stdio.h>

int main() {
   int ms, bs, nob, ef, n, mp[10], tif = 0;
   int i, p = 0;

   printf("Enter the total memory available (in Bytes): ");
   scanf("%d", &ms);

   printf("Enter the block size (in Bytes): ");
   scanf("%d", &bs);

   nob = ms / bs;
   ef = ms - nob * bs;

   printf("\nEnter the number of processes: ");
   scanf("%d", &n);

   printf("\nNo. of Blocks available in memory: %d\n", nob);
   printf("\nPROCESS\tMEMORY REQUIRED\tALLOCATED\tINTERNAL
FRAGMENTATION\n");

   for (i = 0; i < n && p < nob; i++) {
      printf("%d\t\t%d", i + 1, mp[i]);

      if (mp[i] > bs) {
         printf("\t\tNO\t\t---");
      } else {
         printf("\t\tYES\t%d", bs - mp[i]);
         tif = tif + bs - mp[i];
         p++;
      }

      printf("\n");
   }

   if (i < n) {
      printf("Memory is Full, Remaining Processes cannot be accommodated\n");
   }
```

```
    printf("\nTotal Internal Fragmentation is %d\n", tif);
    printf("Total External Fragmentation is %d\n", ef);

    return 0;
}
```

OUTPUT:

15.b. Multiprogramming with A Variable Number Of Tasks (MVT)

```
#include <stdio.h>

int main() {
   int ms, mp[10], i, temp, n = 0;
   char ch = 'y';

   printf("Enter the total memory available (in Bytes): ");
   scanf("%d", &ms);

   temp = ms;

   while (ch == 'y' || ch == 'Y') {
      printf("Enter memory required for process %d (in Bytes): ", n + 1);
      scanf("%d", &mp[n]);

      if (mp[n] <= temp) {
         printf("Memory is allocated for Process %d\n", n + 1);
         temp = temp - mp[n];
         n++;
      } else {
         printf("Memory is Full\n");
         break;
      }

      printf("Do you want to continue (y/n): ");
      scanf(" %c", &ch);
   }

   printf("\nTotal Memory Available: %d\n", ms);
   printf("\n\tPROCESS\t\tMEMORY ALLOCATED\n");

   for (i = 0; i < n; i++) {
      printf("\t%d\t\t%d\n", i + 1, mp[i]);
   }

   printf("\nTotal Memory Allocated is %d\n", ms - temp);
   printf("Total External Fragmentation is %d\n", temp);

   return 0;
}
```

OUTPUT:

Experiment - 16

Write a program to implement first fit, best fit and worst fit algorithm for memory management.

16.a FIRST-FIT

```
#include <stdio.h>
#include <conio.h>
#define max 25
void main() {
   int frag[max], b[max], f[max], i, j, nb, nf, temp, highest = 0;
   static int bf[max], ff[max];

   clrscr();

   printf("\n\tMemory Management Scheme - Worst Fit");
   printf("\nEnter the number of blocks:");
   scanf("%d", &nb);
   printf("Enter the number of files:");
   scanf("%d", &nf);

   printf("\nEnter the size of the blocks:-\n");

   for (i = 1; i <= nb; i++) {
      printf("Block %d:", i);
      scanf("%d", &b[i]);
      bf[i] = 0; // Initialize bf array
   }

   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++) {
      highest = 0; // Reset highest for each file
      for (j = 1; j <= nb; j++) {
         if (bf[j] != 1) { // if bf[j] is not allocated
            temp = b[j] - f[i];
            if (temp >= 0)
               if (highest < temp) {
                  ff[i] = j;
```

```
                highest = temp;
            }
        }
    }
    frag[i] = highest;
    bf[ff[i]] = 1;
}

printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragmentation");

for (i = 1; i <= nf; i++)
    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);

getch();
}
```

OUTPUT:

16.b BEST-FIT

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

#define max 25

void main() {
   int frag[max], b[max], f[max], i, j, nb, nf, temp, lowest = 10000;
   static int bf[max], ff[max];

   clrscr();

   printf("\nEnter the number of blocks:");
   scanf("%d", &nb);

   printf("Enter the number of files:");
   scanf("%d", &nf);

   printf("\nEnter the size of the blocks:\n");
   for (i = 1; i <= nb; i++) {
      printf("Block %d:", i);
      scanf("%d", &b[i]);
      bf[i] = 0; // Initialize bf array
   }

   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++) {
      lowest = 10000; // Reset lowest for each file
      for (j = 1; j <= nb; j++) {
         if (bf[j] != 1) {
            temp = b[j] - f[i];
            if (temp >= 0 && lowest > temp) {
               ff[i] = j;
               lowest = temp;
            }
         }
      }
      frag[i] = lowest;
```

```
        bf[ff[i]] = 1;
    }

    printf("\nFile No\tFile Size\tBlock No\tBlock Size\tFragment");
    for (i = 1; i <= nf && ff[i] != 0; i++) {
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
    }

    getch();
}
```

OUTPUT:

16.c WORST-FIT

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

#define max 25

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

   clrscr();

   printf("\n\tMemory Management Scheme - First Fit");
   printf("\nEnter the number of blocks:");
   scanf("%d", &nb);

   printf("Enter the number of files:");
   scanf("%d", &nf);

   printf("\nEnter 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++) {
      temp = -1; // Reset temp for each file
      for (j = 1; j <= nb; j++) {
         if (bf[j] != 1) {
            if (b[j] >= f[i]) {
               temp = j;
               break;
            }
         }
      }

      if (temp != -1) {
```

```
            frag[i] = b[temp] - f[i];
            bf[temp] = 1;
            ff[i] = temp;
        }
    }

    printf("\nFile No\tFile Size\tBlock No\tFragment");

    for (i = 1; i <= nf; i++) {
        if (ff[i] != 0) {
            printf("\n%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], frag[i]);
        }
    }

    getch();
}
```

OUTPUT:

Experiment – 17

Simulate Bankers Algorithm for Dead Lock Avoidance & Prevention.

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

struct da {
   int max[10], a1[10], need[10], before[10], after[10];
} p[10];

void main() {
   int i, j, k, l, r, n, tot[10], av[10], cn = 0, cz = 0, temp = 0, c = 0;
   clrscr();

   printf("\n ENTER THE NO. OF PROCESSES:");
   scanf("%d", &n);

   printf("\n ENTER THE NO. OF RESOURCES:");
   scanf("%d", &r);

   for(i = 0; i < n; i++) {
      printf("PROCESS %d \n", i + 1);
      for(j = 0; j < r; j++) {
         printf("MAXIMUM VALUE FOR RESOURCE %d:", j + 1);
         scanf("%d", &p[i].max[j]);
      }
      for(j = 0; j < r; j++) {
         printf("ALLOCATED FROM RESOURCE %d:", j + 1);
         scanf("%d", &p[i].a1[j]);
         p[i].need[j] = p[i].max[j] - p[i].a1[j];
      }
   }

   for(i = 0; i < r; i++) {
      printf("ENTER TOTAL VALUE OF RESOURCE %d:", i + 1);
      scanf("%d", &tot[i]);
   }

   for(i = 0; i < r; i++) {
      for(j = 0; j < n; j++)
         temp = temp + p[j].a1[i];
      av[i] = tot[i] - temp;
      temp = 0;
   }
```

```
printf("\n\t RESOURCES ALLOCATED   NEEDED   TOTAL AVAIL");
for(i = 0; i < n; i++) {
   printf("\n P%d \t", i + 1);
   for(j = 0; j < r; j++)
      printf("%d\t", p[i].max[j]);

   printf("\t");
   for(j = 0; j < r; j++)
      printf("%d\t", p[i].a1[j]);

   printf("\t");
   for(j = 0; j < r; j++)
      printf("%d\t", p[i].need[j]);

   printf("\t");
   for(j = 0; j < r; j++) {
      if(i == 0)
         printf("%d\t", tot[j]);
   }

   printf("\t");
   for(j = 0; j < r; j++) {
      if(i == 0)
         printf("%d\t", av[j]);
   }
}

printf("\n\n\t AVAIL BEFORE\t AVAIL AFTER ");
for(l = 0; l < n; l++) {
   for(i = 0; i < n; i++) {
      for(j = 0; j < r; j++) {
         if(p[i].need[j] > av[j])
            cn++;
         if(p[i].max[j] == 0)
            cz++;
      }

      if(cn == 0 && cz != r) {
         for(j = 0; j < r; j++) {
            p[i].before[j] = av[j] - p[i].need[j];
            p[i].after[j] = p[i].before[j] + p[i].max[j];
            av[j] = p[i].after[j];
            p[i].max[j] = 0;
```

```
            }

            printf("\n P %d \t", i + 1);
            for(j = 0; j < r; j++)
               printf("%d\t", p[i].before[j]);

            printf("\t");
            for(j = 0; j < r; j++)
               printf("%d\t", p[i].after[j]);

            cn = 0;
            cz = 0;
            c++;
            break;
         }
         else {
            cn = 0;
            cz = 0;
         }
      }
   }

   if(c == n)
      printf("\n THE ABOVE SEQUENCE IS A SAFE SEQUENCE");
   else
      printf("\n DEADLOCK OCCURRED");

   getch();
}
```

OUTPUT:

Experiment - 18

Simulate The Following Page Replacement Algorithms.

18.a FIFO

```
#include<stdio.h>
void main()
{
int i, j, k, f, pf=0, count=0, rs[25], m[10], n;
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);
}
```

OUTPUT:

18.b LRU

```
#include<stdio.h>
int main()
{
int i, j , k, min,n,f,pf=0;
int rs[25], m[10], count[10], flag[25],next=1;
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);
}
```

OUTPUT:

18.c LFU

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

int fr[10], n, m;

void display();

int main() {
   int i, j, page[20], fs[10];
   int max, found = 0, lg[10], index, k, pf = 0;
   float pr;

   printf("Enter length of the reference string: ");
   scanf("%d", &n);

   printf("Enter the reference string: ");
   for (i = 0; i < n; i++)
      scanf("%d", &page[i]);

   printf("Enter number of frames: ");
   scanf("%d", &m);

   for (i = 0; i < m; i++)
      fr[i] = -1;

   printf("\nReference String\tPage Frames\n");
   for (j = 0; j < n; j++) {
      found = 0;
      for (i = 0; i < m; i++) {
         if (fr[i] == page[j]) {
            found = 1;
            break;
         }
      }

      if (found == 0) {
         int replace_index = -1;
         for (i = 0; i < m; i++) {
            if (fr[i] == -1) {
               replace_index = i;
               break;
            }
```

```
            }

            if (replace_index == -1) {
                int max_dist = -1;
                for (i = 0; i < m; i++) {
                    int dist = 0;
                    for (k = j + 1; k < n; k++) {
                        if (fr[i] == page[k]) {
                            break;
                        }
                        dist++;
                    }
                    if (dist > max_dist) {
                        max_dist = dist;
                        replace_index = i;
                    }
                }
            }

            fr[replace_index] = page[j];
            pf++;
        }

        display();
    }

    printf("Number of page faults: %d\n", pf);
    pr = (float)pf / n * 100;
    printf("Page fault rate: %f\n", pr);

    return 0;
}

void display() {
    int i;
    for (i = 0; i < m; i++)
        printf("%d\t", fr[i]);
    printf("\n");
}
```

OUTPUT:

Experiment - 19

Simulate the Following File Allocation Strategies

19.a Sequenced

```
#include<stdio.h> main()
{
int f[50],i,st,j,len,c,k; clrscr(); for(i=0;i<50;i++) f[i]=0;
X: printf("\n Enter the starting block & length of file");
scanf("%d%d",&st,&len);
for(j=st;j<(st+len);j++) if(f[j]==0)
{
f[j]=1;
printf("\n%d->%d",j,f[j]);
}
else
{
printf("Block already allocated"); break;
}
if(j==(st+len))
printf("\n the file is allocated to disk");
printf("\n if u want to enter more files?(y-1/n-0)"); scanf("%d",&c);
if(c==1) goto X; else exit();
getch();
}
```

OUTPUT:

19.b Indexed

```
#include<stdio.h>
int       f[50],i,k,j,inde[50],n,c,count=0,p; main()
{
clrscr(); for(i=0;i<50;i++) f[i]=0;
x: printf("enter index block\t"); scanf("%d",&p);
if(f[p]==0)
{
f[p]=1;
printf("enter no of files on index\t"); scanf("%d",&n);
}
else
{
printf("Block already allocated\n"); goto x;
}
for(i=0;i<n;i++) scanf("%d",&inde[i]); for(i=0;i<n;i++) if(f[inde[i]]==1)
{
printf("Block already allocated"); goto x;
}
for(j=0;j<n;j++) f[inde[j]]=1;
printf("\n allocated"); printf("\n file indexed"); for(k=0;k<n;k++)
printf("\n %d->%d:%d",p,inde[k],f[inde[k]]);
printf(" Enter 1 to enter more files and 0 to exit\t"); scanf("%d",&c);
if(c==1) goto x; else exit();
getch();
}
```

OUTPUT:

19.c Linked

```
#include<stdio.h> main()
{
int f[50],p,i,j,k,a,st,len,n,c; clrscr();
for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks that are already allocated");
scanf("%d",&p);
printf("\nEnter the blocks no.s that are already allocated");
for(i=0;i<p;i++)
{
scanf("%d",&a); f[a]=1;
}
X: printf("Enter the starting index block & length");
scanf("%d%d",&st,&len);
k=len;
for(j=st;j<(k+st);j++)
{
if(f[j]==0)
{
f[j]=1;
printf("\n%d->%d",j,f[j]);
}
else
{
printf("\n %d->file is already allocated",j);
k++;
}
}
printf("\n If u want to enter one more file? (yes-1/no-0)");
scanf("%d",&c); if(c==1)
goto X;
else exit();
getch( );
}
```

OUTPUT: