Experiment -01

Aim:- To Study the basics of Unix/Linux commands

Description:

a) date:
- used to check the date and time.

Syntax: $ date.

b) cal:
- used to display the calendar

Syntax: $ cal 2 2009

c) echo:
- used to print the message on the screen.

Syntax: $ echo text

d) ls:
- used to list the files in your files are kept in a directory

Syntax:

$ ls
All the files
$ ls -l (provide file statistics)
$ ls -t order by creation time
$ ls -u sort by access time (or show when last accessed together with -l)
$ ls -S order by size
$ ls -r Reverse order
$ ls -f Mark directories with/executable with *, symbolic links with @. local sockets with=, named pipe (FIFO s) with
$ ls -s show file size

$ls -h "Human Readable", Show file size in Kilobytes & Mega bytes (h can be used togethes with (-l or)

$ls -[a-m]* List all the files whoose name begin with alphabets from 'a' to 'm'.

$ls [a]* List all the files whoose name begins begins 'a' or 'A'

Eg: $ls > mylist output 'ls' command is stored to disk file named 'mylist'.

e) lp
—used to take printouts.

Syntax:
$lp filename.

f) man
— used to provide manual help on every UNIX commands.

Syntax:
$man unix command.

$man cat.

File Manipulation Commands:

a) cat- this create, view and concatenate files

Creation

Syntax: $ cat > filename

Viewing

Syntax: $ cat filename

Add text to an existing file:

Syntax: $ cat >> filename

concatenate

Syntax: $ cat file1 file2 > file3

$ cat file1 file2 >> file3 (no over writing of file3)

b) grep- used to search a particular word (or) pattern related to that word from the file.

Syntax: $ grep search word filename

Eg: $ grep search word filename

$ grep anu student

c) rm - deletes a file from the file system

Syntax: $ rm filename

d) touch: - used to create a blank file

Syntax: $ touch filenames

e) cp - copies the files (or) directories

Syntax: $ cp source file destination file

Eg: $ cp student stud

f) mv - to rename the file (or) directory
Syntax: $mv Oldfile newfile
Eg: $miv -i Student Studentlist (-i prompt when overwrite)

g) Cut - it cuts (or) pickup a given number of character (or) fields of the file.
Syntax: $ cut <option> <filename>
Eg: $ cut -c filename
$cut -c1 -10 emp
$cut -f 3,6 emp
$ cut -f 3-6 emp
-c cutting columns.
-f cutting fields.

h) head - displays 10lines from the head (top) of a given file
Syntax: $ head filename
Eg: $ head Student
To display the top two lines:
Syntax: $ head -2 Student

i) tail - displays last 10 lines of the file
Syntax: $tail filename
Eg: $tail Student
To display the bottom two lines;
Syntax: $tail -2 Student

GIT

j) chmod - used to change the permissions of a file or directory

Syntax: $ chmod category operation permissionfile

where, Category - is the usertype

Operation - is used to assign or remove permission.

File - are used to assign or remove permission all.

Examples:

$ chmod u - wx Student

Removes write and execute permission for users

$ chmodu +rw, g+rw Student

Assigns read and write permission for users and groups.

$ chmod g=rwx Student

Assigns absolute permission for group of all read, write and execute permissions.

Aim:- Write a C program using the I/O System calls of Unix/Linux Operating System (Open, read, write, close)

program:

```
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
int main()
{
 int n, i=0;
 int f1, f2;
 char c, string[100];
 f1= open("data", O_RDWR|O_CREAT|O_TRUNC);
 while((c= getchar())!='\n')
 {
  string[i++] = c;
 }
  string[i] = '\0';
 write (f1, string, i);
 close (f1);
 f2 = open ("data", O_RDONLY );
 read (f2, string, 0);
 printf("\n %s \n", string);
 close(f2);
 return 0;
}
```

Output:

Rakesh

Rakesh

Aim:- Write a C program using Opendir(), Closedir(), readdir().

Program:

```
#include < stdio.h>
# include < fcntl.h>
# include < dirent.h>
main()
{
char d[10];
int c, op;
DIR *e;
Struct dirent * sd;
printf (" ** menu \n 1. create dir \n 2. remove
   dir \n 3. read dir \n");
printf(" enter your choice:\n");
Scanf(" %d", &op);
Switch(op)
{
Case 1;
   printf(" enter dir name:\n");
   Scanf(" %s", & d);
   c= mkdir (d, 777);
   if (c==1)
      printf(" dir is not created ");
   else
      printf(" directory is Created ");
   break;
```

```
case 2:
    printf("enter dir name \n");
    scanf("%s", &d);
    c= rmdir(d);
    if (c==1)
     printf(" dir is not removed ");
    else
     printf(" dir is removed ");
     break;

case 3:
    printf("enter dir name to open");
    scanf("%s", &d);
    e= opendir(d);
    if (e==NULL)
     printf(" dir does not exist");
    else
     {
      printf("dir exist \n");
    while((sd=readdir(e))!=NULL)
      printf("%s\t", sd→ d_name );
     }
     closedir(e);
     break;
  }
}
```

Output:

```
** menu
1. create dir
2. remove dir
3. read dir

enter your choice: 1
enter dir name : csm
directory is created.
```

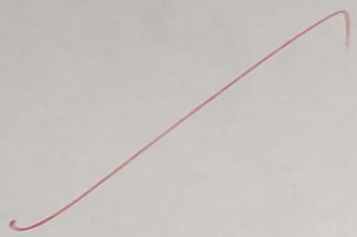
Aim: Write a program to implement fork system call.

Program:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
void main()
{
int pid, n;
pid= fork();
if (pid ==0)
{
printf(" child process and its pid is %d \n",
                                    getpid());
printf(" parent process and its pid is %d \n",
                                    getpid());
}
if (pid >0)
{
  printf(" parent process and its pid is %d \n",
                                    getpid());
  printf(" child process and its pid is %d \n",
                                    pid);
}
}
```

Output:

Parent process and its pid is 8058
Child process and its pid is 8059
Child process and its pid is 8059
parent process and its pid is 8058



Aim: Write a program to implement waitpid.

Program:

```
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
int main(int argc, char* argcv[])
{
   pid_t cpid, w;
   int status;
   cpid = fork();
   if (cpid==-1)
   {
    perror("fork"); exit(EXIT_FAILURE); }
    if (cpid==0)   // code executed by child
   {
    printf("Child PID is %ld \n",(long)getpid());
     if(argc==1)
       pause();
       - exit(atoi(argv[1]));
   }
   else
   {
     do
     {
       w= waitpid (cpid, &status, WUNTRACED|
                                  WCONTINUED);
```

```
if(w==-1)
{
perror("wait pid"); exit(EXIT_FAILURE);}
if(WIFEXITED(status))
 {
  printf("exit, status = %d\n",
                    WEXITSTATUS(status));
 }
else if(WIFSIGNALED(status))
 {
   printf("Killed by signal %d\n",
                    WTERMSIG(status));
 }
elseif(WIFSTOPPED(status))
 {
  printf("stopped by signal %d\n",
                    WSTOPSIG(status));
 }
 elseif(WIFCONTINUED(status))
 {
   printf("continued\n");
 }
}
while(!WIFEXITED(status) && !WIFSIGNALED
                    (status));
 exit(EXIT_SUCCESS);
}
}
```

Output:

```
./a.out &
[1] 2424
$ child pid is 2425
  Kill - STOP 2425
$ Stopped by Signal 19
  ps
   PID    TTY        TIME        CMD
  2315    pts/1    00:00:00      bash
  2425    pts/1    00:00:00      a.out
  2425    pts/1    00:00:00      a.out
  2428    pts/1    00:00:00      ps
$ Kill -CONT 2425
   Continued
$ Kill -TERM 2425
   Killed by Signal 15
```

Aim: Write a program to implement File IO

Program:

```
#include <stdio.h>
#include<unistd.h>
#include< fcntl.h>
void main()
{
 char but;
 int fd1,fd2
 fd1 = open ("first-file",O_RDONLY|O_CREAT);
 if(fd1 ==-1)
 {
 Printf(" Error opening First_file \n");
  close(fd1);
  return;
 }
 fd2 = open ("second-file", O_WRONLY|O_CREAT);
 while (read (fd1, &but ,1))
  {
    write (fd2 ,&but , 1);
  }
 printf(" Succesful copy ");
 close (fd1);
 close( fd2);
}
```

Output: Successful copy

Aim: Write a shell program to check whether the given number is even (or) odd.

Program:

```
#!/bin/bash
echo "Enter a number: "
read n
rem=$(( $n % 2 ))
if [ $rem -eq 0 ]
then
   echo" Number is even"
else
   echo" Number is odd "
fi
```

Output:

```
Enter a number
5
Number is odd
```

Aim: Write a shell program to check whether the given year is leap (or) not.

Program:

```
#! /bin/bash
echo "enter the year"
read y
n= $(( $y % 4))
if [ $n -eq 0 ]
then
    echo "its a leap years"
else
    echo "not a leap year"
fi
```

Output:

```
enter the years
2016
its a leap years
```

Aim: Write a shell program to swap two numbers using a third variable

Program:

```
echo " Enter Two Numbers"
read a b
temp = $a
a= $ b
b= $temp
echo " after swapping"
echo $a $b
```

Output:

```
Enter Two Numbers
 5  6
after swapping
 6  5
```

Aim: Write a C program to implement First comes First Serves CPU scheduling Algorithm.

program:

```
#include <stdio.h>
main()
{
int bt[20], wt[20], tat[20], i, n;
float wtavg, tatavg;
printf("\n Enter the number of processes--");
scanf("%d", &n);
for(i=0; i<n; i++)
{
printf("\nEnter the Burst time for process %d
                                -- ", i);
 scanf("%d", &bt[i]);
}
wt[0]= wtavg= 0;
tat[0]= tatavg= bt[0];
for(i=1; i<n; i++)
{
 wt[i]= wt[i-1] + bt[i-1];
 tat[i]= tat[i-1] + bt[i];
 wtavg = wtavg +wt[i];
 tatavg= tatavg +tat[i];
}
printf("\t process \t Burst Time \t Waiting Time
 \t Turn Around Time\n");
```

```
for(i=0; i<n; i++)
printf("\n\t p%d\t\t %d \t\t %d \t\t %d \n",
        i, bt[i], wt[i], tat[i]);
printf("\n Average Waiting Time -- %f ",
                        wtavg/n);
printf("\n Average Turnaround Time -- %f",
                        tatavg/n);
}.
```

Output:

```
enter the number of processes --3
enter the burst process time for process 0
12
enter the burst process time for process 1
34
enter the burst process time for process 2
3
```

Process	Burst Time	Waiting Time	TurnAround Time
P0	12	0	12
P1	34	12	46
P2	3	46	49

Average Waiting Time --19.333334
Average Turnaround Time --35.666668

Aim: Write a C program to implement Shortest Job first CPU Scheduling algorithm.

Program:

```
#include <stdio.h>
void main()
{
int P[20], bt[20], wt[20], tat[20], i, k, n, temp;
float wtavg, tatavg;
printf("\n Enter the number of processes --");
scanf("%d", &n);
for(i=0; i<n; i++)
{
 P[i] = i;
 printf("Enter Burst Time for process %d", i);
 scanf("%d", &bt[i]);
}
for(i=0; i<n; i++)
 for(k=i+1; k<n; k++)
    if (bt[i] > bt[k])
    {
      temp = bt[i];
      bt[i] = bt[k];
      bt[k] = temp;
      temp = P[i];
      P[i] = P[k];
      P[k] = temp;
}
```

```
wt[0] = wtavg=0 ;
tat[0]= tatavg = bt[0];
for (i=1; i<n;i++)
 {
   wt[i] = wt[i-1] +bt[i-1];
   tat[i] = tat[i-1] + bt[i];
   wtavg= wtavg + wt[i];
   tatavg = tatavg + tat[i];
 }
printf("\t process \t Burst Time \t waiting Time \t
  Turn Around Time \n");
  for(i=0; i<n; i++)
   printf("\n\t P%d \t\t %d \t\t %d \t\t %d\n
      ", i, bt[i], wt[i], tat[i]);
printf("\n Average Waiting Time -- %f",
                          wtavg/n);
printf("\n Average Turn Around Time -- %f",
                          tatavg/n);
}
```

Output!

enter the number of processes -- 3
enter the burst time for process 0
12
enter the burst time for process 1
3
enter the burst time for process 2
4

Process	Burst Time	Waiting Time	Turn Around Time
P0	3	0	3
P1	4	3	7
P2	12	7	19.

Average waiting time -- 3.333333

Average turn Around Time -- 9.666666

Aim: Write a C program to implement priority-scheduling algorithm.

Program:

```
#include<stdio.h>
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
        processes %d --", i);
 scanf("%d %d", &bt[i], &pri[i]);
}
for(i=0; i<n; i++)
  for( k=i+1; k<n; k++)
    if(pri[i] > pri[k])
        temp = p[i];
        p[i] = p[k];
        p[k] = temp;
        temp = bt[i];
        bt[i] = bt[k];
        bt[k] = temp;
```

```
        temp= pri[i];
        pri[i]= pri[k];
        pri[k]= temp;
    }
    wtavg= wt[0]=0;
    tatavg= tat[0]= bt[0];
  for(i=1; i<n; i++)
  {
    wt[i]= wt[i-1] + bt[i-1];
    tat[i]= tat[i-1] + bt[i];
    wtavg= wtavg + wt[i];
    tat avg= tat avg+ 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\n",
                i, bt[i], wt[i], tat[i]);
printf("\n Average Waiting Time --- %f",
                wtavg/n);
printf("\n Average Turnaround Time --- %f",
                tat avg/n);
}
```

output:

enter the number of processes --- 3

enter the burst time for process 0

12

2

enter the burst time for process 1

3

4

enter the burst time for process 2

5

1

process	Burst Time	Waiting Time	TurnAround Time
P0	5	0	5
P1	12	5	17
P2	3	17	20

Average waiting Time --- 7.333333

Average TurnAround Time --- 14.000000

Aim: Write a C program to implement Round Robin CPU scheduling algorithm.

Program:

```
#include <stdio.h>
main()
{
 int i,j,n,bt[10], wt[10], tat[10], t, ct[10],
   max;
 float awt=0, att=0, temp=0;
 printf(" Enter the no. of process: ");
  scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  printf("\n Enter Burst time for process %d:"
                                    i+1);
  scanf("%d", &bt[i]);
  ct[i]= bt[i];
 }
printf("\n Enter the size of slice : ");
 scanf("%d", &t);
 max = bt[0];
 for (i=0; i<n; i++)
   if(max < bt[i])
     max = bt[i];
  for(j=0; j< (max/t) +1; j++)
   for(i=0; i<n; i++)
    if(bt[i] != 0)
```

```
if (bt[i]<=t)
{
 tat[i] = temp + bt[i];
  temp= temp+bt[i];
  bt[i] = 0;
}
else
 {
  bt[i] = bt[i]-t;
  temp= temp+t;
 }
for(i=0; i<n; i++)
{
 wt[i]= tat[i]-ct[i];
  att = att+ tat[i];
  awt = awt+wt[i];
 }
printf("\n the average turn around
   time is : %f", att/n);
printf("\n the average waiting Time is :
          %f ", awt/n);
printf("\n\t process \t Burst Time \t waiting
   Time \t Turn Around Time \n");
for(i= 0; i<n; i++)
printf("\t %d \t %d \t\t %d \t\t %d \n", i+1,
        ct[i], wt[i], tat[i]);
}
```

Enter the no. of processes : 3

Enter Burst Time for process 1 : 24

Enter Burst Time for process 2 : 3

Enter Burst Time for process 3 : 3

enter time Slice : 4

Average TurnAround Time : 15.666667

Average Waiting Time : 5.666667

Process	BurstTime	WaitingTime	Turnaround Time
1	24	6	30
2	3	4	7
3	3	7	10

Algorithm For First comes First Serves (FCFS).

Step-1: Start

Step-2: Declare the variable i, n as integer and bt, wt, tat of array size 20.

Step-3: declare the float variables wtavg, tatavg.

Step-4: Get the value of n assign process id with burst time

Step-5: Assign wt[0] and wtavg as Zero.

tat[0] = tatavg = bt[0]

Step-6: Calculate the waiting Time and Turn around Time using loop.

Step-7: Calculate the average waiting Time and average turn around Time by dividing total no. of process & print them.

Step-8: Stop.

Algorithm for Shortest Job First (SJF).

Step-1: Start

Step-2: declare the variables i, k, n, P[20], bt[20], wt[20], as integer

Step-3: declare float variables wtavg, tatavg.

Step-4: Get the value of n and assign the burst times into process id.

Step-5: enter the burst times according to values, the least value should enter first into process stack (using swapping)

Step-6: Calculate avg waiting time and avg turn around time by dividing waiting time & tat by no. of process.

Step-7: Stop.

Algorithm for priority scheduling.

Step-1: Start

Step-2: declare the variables and enter the limit of processes

Step-3: enter the burst times into the array with priority

Step-4: change the process id by swapping according to the priority of processes.

Step-5: Calculate waiting Time & Turn around Time by dividing no. of process.

Step-6: print average waiting Time & average Turn around Time.

Step-7: Stop

Algorithm for Round Robin (RR):

Step-1: Start

Step-2: declare the variables and enter the limit of processes.

Step-3: enter the processes burst time into array. take time Quantum also.

Step-4: using Time Quantum calculate Turnaround Time {using tat & bt calculate waiting Time (wt=tat-bt)

Step-5: Calculate the average waiting Time & Turn around Time and print them.

Step-6: Stop

Aim: Write a c program to implement IPC mechanism pipe.

Program:

```
#include < stdio.h>
#include <unistd.h>
#include < sys/types.h>
#include < sys/wait.h>
void main()
{
  int fd[2], n;
  char buffer[100];
  pid_t  p;
  pipe(fd);
  p= fork();
  if (p>0)
  {
   printf("parent passing into to child\n");
   write(fd[1], "pipe concept\n", 14);
   wait(10);
  }
 else
 {
  printf("child printing received value\n");
   n= read(fd[0], buffer, 100);
   write(1, buffer, n)
 }
}
```

Output!

Parent passing into to child
Child printing received value
Pipe concept.

Aim: Write a c program on IPC Mechanisms using fifo pipe.

Three stages are involved in implementing the concept.

Step-1: Create a named pipe.
Step-2: Sender process write the data into the pipe.
Step-3: Receiver process will read the data from the pipe.

Program:

```
// To create a Named Pipe.
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
{
 void main()
  {
   int res;
   mkfifo ("fifo1", 0777);
   printf("Named pipe created \n");
  }
// writing to fifo pipe
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
```

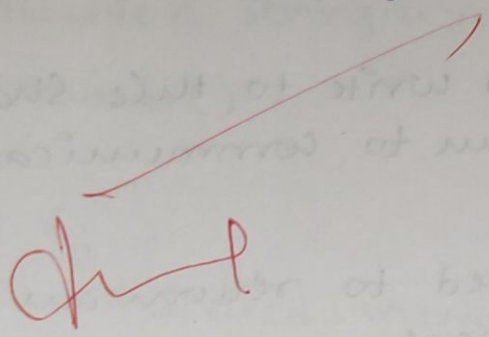
```
void main()
{
 int res,n;
 res = open("fifo1", O_WRONLY);
 write(res, "Hello", 4);
 printf("Sender process %d Send the data
                                \n",getpid());
}
```

// Reading from the fifo name

```
#include <unistd.h>
#include< stdio.h>
#include< fcntl.h>
void main()
{
 int res,n;
 char buffer[100];
 res= open("fifo1", O_RDONLY);
  n= read(res, buffer,100);
 printf(" Reader process %d  Started \n",
                                getpid());
 printf(" data received by receiver %d is
          %s\n", getpid(), buffer);
}
```

Output:

named pipe created
Sender process 2695 , sent the data
reader process 2683 started
data received by receiver 2683 is hello



Aim:- Write a C program to implement Shared Memory (writer process).

Algorithm:

- Shared memory is an IPC Mechanism where multiple processes can access the same region of memory to share data.
- The processes can read and write to this shared memory area, allowing them to communicate efficiently.
- Shmget is a SystemCall used to return an identifier for the shared memory segment

  Shmget (Key_t, Size_tsize, int shmflg);
- Shmat is a System call used to attach identifier to shared memory address.

  Shmat(int shmid, void* shmaddr, int shmflg);
- read the buffer which is entered by the writer process.
- Strcpy is used to copy the set of string from one location to another.
- print the string which is written by the writer process.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>
#include <string.h>
int main()
{
 int i;
 void *shared_memory;
 char buffer[100];
 int shmid;
 shmid = shmget((key_t)2345, 1024, 0666|IPC_CREAT);
 printf("key of shared memory is %d \n", shmid);
 shared_memory = shmat(shmid, NULL, 0);
// process attached to at shared memory segment.
 printf("process attached at %p \n", shared_memory);
// This prints the address where the segment is attached
  with this process.
 printf("enter some data to write to shared memory
                                        \n");
 read(0, buffer, 100); // get some input from user.
 strcpy(shared_memory, buffer);
 printf("you wrote: %s \n", (char *) shared_memory);
}
```

Output:

```
Key of Sharedmemory is 1605641
process attached at 0x7f77c19e3000
enter some data to write to Shared Memory
this is rakesh.
you wrote: this is rakesh.
```

Output:

```
Key of Sharedmemory is 1605641
process attached at 0x7f77c19e3000
enter some data to write to Shared Memory
this is rakesh.
you wrote: this is rakesh.
```

Aim: Write a C program to implement Shared Memory (reader process)

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
int main()
{
 int i;
 void * shared_memory;
 char buffer[100];
 int shmid;
 shmid= shmget((key_t)2345, 1024, 0666);
 printf("Key of shared memory is %d\n", shmid);
shared_memory = shmat(shmid, NULL, 0);
//process attached at shared Memory segment
 printf("process attached at %p\n", shared_memory);
printf("data read from shared memory is %s\n",
                    (char *) shared_memory);
}
```

Output:

```
Key of shared memory is 160541
process attached at  0x7f311cd0a000
data read from shared memory is: this is rakesh.
```

Aim:- Write a shell program to concatenate two strings.

program:

Step-1: Start the program.
Step-2: Enter into nano editor for entering the code.
Step-3: Read the First string.
Step-4: Read the second string.
Step-5: concatenate the two strings.
Step-6: Enter into the Escape mode for the Execution of Result and verify the Output.
Step-7: Stop the program.

```
echo " enter the First String"
read str1
echo " enter the second string "
read str2
echo " the Concatenated string is " $str1 $str2
```

output:

Enter the First string : Hello
enter the second string : World.
The Concatenated string is HelloWorld.

Aim: Write a shell program to compare the two strings

Algorithm:

Step-1: Start the program.

Step-2: Enter into the vi editor and go to insert mode for entering the code.

Step-3: Read the first string.

Step-4: Read the second string.

Step-5: Compare the two strings using the if loop

Step-6: If the condition satisfies print that two strings are equal. else print two strings are not equal.

Step-7: Enter into the escape mode for execution of the result and verify the output.

Step-8: Stop the program.

Program:

```
echo "enter a string1"
read first
echo "enter a string2"
read second
if [$first = $second]# this"=" for other than digits equal
then
echo " strings are equal "
else
 echo" strings are unequal"
fi
```

Output:

```
enter a string1
abcd
enter a string2
abcd
 strings are equal.
```

Aim:- Write a shell program to compare of two strings from commandline.

Algorithm:

Step-1: Start the program.
Step-2: Enter into the vi editor and go to the insert mode for entering the code
Step-3: Read the First string
Step-4: Read the second string
Step-5: Compare the two strings using the if loop.
Step-6: If condition satisfies then print that two strings are equal else print two strings are not equal.
Step-7: Enter into the Escape mode for execution of the Result and verify the output.
Step-8: Stop the program.

Program:

~~echo " enter a string1 "~~
~~read first~~
~~echo " enter a string2 "~~
~~read second.~~
~~if [$first $second]~~
~~fi [$first~~

```
if [ $1 = $2 ]
then
echo " Strings are equal"
else
echo " String are not equal"
fi
```

Output:

```
chmod +x compare.sh
./compare.sh rakesh rakesh
Strings are equal.
```

Aim: Write a c program to implement message Queue (Sender process).

Algorithm:

- A message Queue is an Interprocess communication mechanism that allows processes (cooperative) to exchange data in the form of messages between two (or) more processes.
- To perform communication using message Queues the following steps are followed

Step-1: create a message Queue (or) connect to an already existing Queue using msgget() system call.

Syntax:

```
int msgget(Key_t Key, int msgflg);
```

The First parameter is used to name a message Queue in system.
Second parameter is permission, Ex: IPC_CREAT.

Step-2: Write the message (or) Send the message into the message Queue using msgsnd() System call.

Syntax:

```
int msgsnd(int msgid, const void *msg_ptr, size_t msg_sz, int msgflg);
```

Step-3: Read the message from the message Queue using msgrcv() system call.

Syntax:

```
int msgrcv(int msgid, const void *msg_ptr, size_t msg_sz, long int msgtype, int msgflg);
```

Step-4: performing control on message Queue using msgctl() System call, we use this system call for deleting message Queue that has been created.

Syntax:

```
int msgctl(int msgid, int command, struct msg_ds *buf);
```

Program:

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
struct msg_buffer
{
  long msg_type;
  char msg[100];
} message;
void main()
{
  key_t my_key;
  int msg_id;
  my_key = ftok("linux", 66);
  msg_id = msgget(my_key, 0666 | IPC_CREAT);
  message.msg_type = 1;
  printf("u write message: ");
  fgets(message.msg, 100, stdin);
  msgsnd(msg_id, &message, sizeof(message), 0);
  printf("Sent message is : %s \n", message.msg);
}
```

Output:

```
Write message: rakesh from rngit
Sent message: rakesh from rngit
```

Aim: Write a C program to implement message Queue (receiver process).

Program:

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

struct msg_buffer
{
 long msg_type;
 char msg[100];
} message;
void main()
{
 key_t my_key;
 int msg_id;
 my_key = ftok("linux", 66);
 msg_id = msgget (my_key, 0666 | IPC_CREAT);
msgrcv(msg_id, &message, sizeof(message),0,1);
 printf("received message is : %s \n",
                        message.msg);
}
```

Output:

```
received message is : rakesh from mgit.
```

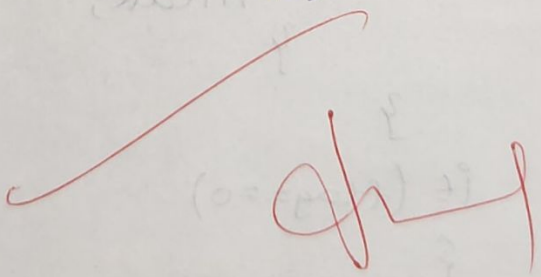
Aim: Write a c program to implement Bankers Algorithm.

Program:

```
#include<stdio.h>
int main()
{
 // P0, P1, P2, P3, P4 are the process names here.
int n,m,i,k,j;
n=5;      // number of processes
m=3;      // number of resources
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};  //available Resources.
int f[n],ans[n],ind=0;
for(k=0;k<n;k++)
 {
  f[k]=0;
 }
int need[n][m];
for(i=0;i<n;i++)
 {
    for(j=0;j<m;j++)
     {
```

```
    need[i][j] = max[i][j] - alloc[i][j];
  }
int y=0;
for(k=0;k<5;k++)
 {
   for(i=0; i<n; i++)
    {
      if(f[i]==0)
      {
        int flag=0;
          for(j=0 ; j<m ; j++)
           {
              if (need[i][j] > avail[j])
               {
                 flag=1;
                 break;
               }
           }
        if (flag==0)
        {
          ans[ind++]=i;
           for(y=0; y<m ;y++)
             avail[y]+ = alloc[i][y];
           f[i]=1;
        }
    }
 }
}
```

```
int flag=1;
for(int i=0; i<n; i++)
{
  if(f[i] ==0)
  {
    flag= 0;
    printf("The Following system is not safe");
    break;
  }
}
if(flag==1)
{
  printf("Following is the SAFE sequence \n");
  for(i=0; i<n-1; i++)
    printf(" P%d →", ans[i]);
  printf("P%d", ans[n-1]);
}
return(0);
}
```



Output!

Following is the SAFE sequence

$P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$.

Aim:- Write a c program to implement Bankers algorithm (dynamic).

Data Structures:-

n-Number of processes, m- number of resource types

Available: Available[j]=k, k-instance of resource type Rj is available

Max: if max[i,j]=k, Pi may request atmost k instances of Rj.

Allocation: If Allocation [i,j]=k, Pi may allocted to k instances of resources.

RjNeed: if Need [I,j]=k, Pi may need k more instances of resource type

Rj Need[Ij] = Max[Ij]- Allocation [I,j];

Safety Algorithm:-

1. Work and Finish be the vector of length m and n respectively

   Work= Available and Finish[i]=False.

2. Find an i such that both

   Finish[i]=False

   Need<=Work

   If no such I exists go to step4.

3. Work= Work + Allocation, Finish[i]=true;

4. If Finish[I]= True for all I, then the System is in Safe State.

Resource Request Algorithm:

Let Request i be Request vector for the process Pi, if request i * [j] = k, then process Pi wants k instances of resource type Rj.

1. if Requesti <= Needi go to step 2
   otherwise Raise an error since process has exceeded its maximum claim.
2. if Requesti ≤ Available go to step-3
   otherwise, P must wait since the Resources are not available
3. The System pretends to have allocated the Requested Resources to process Pi by modifying the state as well

   Available = Available − Requesti
   Allocation = Allocation i + Requesti
   Needi = Needi − Requesti

Algorithm:-

1. Start
2. Get the values of Resources and processes
3. Get the avail value
4. After allocation find the Need value
5. Check whether its possible to allocate
6. If it is possible then system is in safestate.
7. Else system is not in safestate
8. If new request comes then check that system is in safety.
9. or not if we allow the Request.
10. Stop

-Program:-

```
#include<stdio.h>
#include<conio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
 int i,j;
 printf("\n*** Banker's Algo***\n");
 input();
 show();
 cal();
 getch();
 return 0;
}
void input()
{
 int i, j;
 printf("Enter the no. of process\t");
 scanf("%d",&n);
 printf("Enter the no. of resources instances\t");
 scanf("%d",&r);
```

```
printf(" Enter the Max Matrix \n");
 for(i=0; i<n; i++)
 {
  for(j=0; j<r; j++)
   {
    scanf("%d", &max[i][j]);
   }
 }
printf(" Enter the Allocation Matrix \n");
  for(i=0; i<n; i++)
  {
   for(j=0; j<r; j++)
    {
     scanf(" %d", &alloc[i][j]);
    }
  }
printf(" Enter the available Resources \n");
   for(j=0; j<r; j++)
   {
    scanf(" %d", &avai[j]);
   }

}

void show()
 {
  int i, j;
  printf(" process\t Allocation \t Max \t Available \t");
   for(i=0; i<n; i++)
   {
    printf("\n P%d \t", i+1);
     for(j=0 ; j<r ; j++)
      {
```

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

void cal()
{
 int finish[100], temp, need[100][100], flag=1, k, c1=0;
 int safe[100];
 int i, j;
 for(i=0; i<n; i++)
 {
  finish[i]= 0;
 }
 // find need Matrix
   for(i=0; i<n; i++)
    {
      for(j=0; j<r; j++)
       {
        need[i][j]= max[i][j] - alloc[i][j];
       }
    }
```

```
printf("\n");
while(flag)
{
 flag=0;
 for(i=0;i<n;i++)
  {
   int c=0;
   for(j=0;j<r;j++)
    {
     if((finish[i]==0)&&(need[i][j]<=avail[j]))
      {
       c++;
       if(c==r)
       {
         for(k=0;k<r;k++)
         {
          avail[k]+=alloc[i][j];
          finish[i]=1;
          flag=1;
         }
       printf("P%d->",i);
       if(finish[i]==1)
       {
         i=n;
       }
      }
    }
  }
}
```

```
for(i=0; i<n; i++)
 {
  if(finish[i]==1)
  {
   a++;
  }
  else
  {
   printf("P%d->", i);
  }
 }
 if(c1==n)
 {
  printf("\n The system is in safestate");
 }
 else
 {
  printf("\n processes are in deadlock");
  printf("\n System is in unsafestate");
 }
}
```

Output:-

```
Enter the no. of processes 5
Enter the no of resources instances 3
Enter the max matrix
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
```

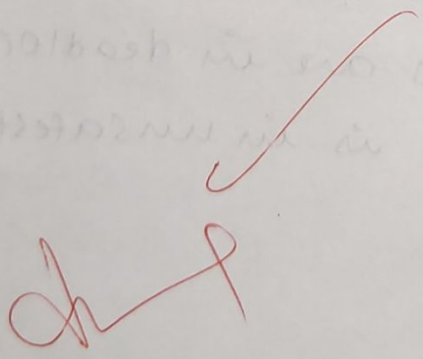
Enter the Allocation Matrix

0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter available resources 3 3 2

$P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_2 \rightarrow P_0$

The system is in safe state.



Aim:- Write a C program to implement producer consumer problem.

Program:-

```
#include <stdio.h>
#include <stdlib.h>
int mutex=1;
int full=0;
int empty=3, data=0;
void producer()
{
  --mutex;
  ++full;
  --empty;
  data++;
  printf("\n producer produces item number:
                                %d\n",data);
  ++mutex;
}
void consumer()
{
  --mutex;
  --full;
  ++empty;
  printf("\n consumer consumes item
          number: %d\n",data);
  data--;
  ++mutex;
}
```

```
int main()
{
 int n, i;
 printf("\n 1. Enter 1 for producer\n");
 printf("\n 2. Enter 2 for consumer\n");
 printf("\n 3. Enter 3 for EXIT\n");

    for(i=1; i>0; i++)
    {
     printf("\n Enter your choice: ");
     scanf("%d", &n);
     switch(n)
     {
      case 1:
         if((mutex==1) && (empty!=0))
          {
           producer();
          }
         else
         {
          printf("The Buffer is full. New
           data cannot be produced!");
         }
         break;
        case 2:
         if ((mutex==1)&&(full!=0))
          {
           consumer();
          }
```

```
        else
        {
         printf("The Buffer is empty! new data cannot
                 be consumed");
        }
        break;
      case 3:
           exit(0);
           break;
      }
  }
}
```

Output:-

1. enter 1 for producer.
2. enter 2 for consumer.
3. enter 3 for Exit.

enter your choice: 1
producer produces item: 1
enter your choice: 1
producer produces item: 2
enter your choice: 1
producer produces item: 3
enter your choice: 1
the buffer is full, new data cannot be produced
enter your choice: 2
consumer ~~produces~~ consumes item number: 3
enter your choice: 2
consumer consumes item number: 2
enter your choice: 3

Aim:- Write a C program to implement paging Technique of memory management.

Program:

```
#include <stdio.h>
int main()
{
int ms, mp, framesize, noframes, freeFrames,
    phyadd, pgno;
int i, j, x, y, offset;
int procesReq[10]={0}, pageTable[10][20]={0};
printf("Enter the memory size\n");
scanf("%d", &ms);
printf("Enter the size of the frames (or) page\n");
scanf("%d", &framesize);
noframes = ms/framesize;
printf("The number of frames available is %d\n",
                                    noframes);
printf("Enter the number of processes\n");
scanf("%d", &np);
freeFrames = noframes;
for(i=1; i<=np; i++)
{
  printf("Enter no. of frames required for
                process %d\n", i);
  scanf("%d", &ProcesReq[i]);
```

```
        if(procesReq[i] > freeFrames)
        {
         printf("Memory is full \n");
         break;
        }
      freeFrames = freeFrames - procesReq[i];
    printf("Enter the page table data for the proces %d:-",
                                                    i);
    for(j=1; j<=procesReq[i];j++)
      scanf("%d", &pageTable[i][j]);
}
printf("Enter the logical address to find the physical
        address\n");
printf("Enter the process no, page no and offset value\n");
  scanf("%d %d %d", &x, &y, &offset);
  if(x > np)
  {
   printf("Invalid proces no");
   printf("Number of processes in the system is %d\n",
                                                    np);
  }
   else if(y >= procesReq[x])
   {
     printf("invalid page no\n");
     printf("proces %d has %d pages\n", x, procesReq[i]);
   }
```

```
else if(offset >= framesize)
{
 printf(" invalid offset value \n");
 printf(" offset value should be less than or equal
      to %d \n", framesize);
}
else
{
 phyadd = pageTable[x][y] * framesize + offset
 printf("\n the physical address is -- %d ", phyadd);
}
 return 0;
}
```

Output:

```
Enter the memory size
100
Enter the size of the frame (a) page
10
The number of frames available is 10
Enter the number of processes
1
Enter no of frames required for process1
4
Enter the pagetable data for the process 1---
2
4
7
9
Enter the logical address to find the physical address.
Enter the process no, page no and offset value.
1 3 2
the physical address is --72.
```

Aim:- Write a C program for the Implementation of Segmentation technique of Memory Management.

Description:

Segmentation is a memory Management Technique that Supports the user view of memory. A logical address space is a collection of segments. Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment. The user therefore specifies each address by two quantities: a segment name and an offset.

Segment table contains mainly two information about segment.

1. Base: It is the base address of the segment.
2. Limit: It is the length of the segment.

The segment number is mapped to segment table the limit of respective segment is compared with offset. if the offset is less than the limit then the address is valid otherwise it throws an error as the address is invalid.

In the case of valid address, the base address of the segment is added to the offset to get the physical address of actual word in the main memory.

program:

```
#include <stdio.h>
int main()
{
  int SegTable [10][2] , Numseg , baseAdd, Size,
      offset, segno , phyAdd;
  int i;
  printf(" Enter the number of segments of
                process \n");
  scanf("%d", &Numseg);
  for(i=0; i< Numseg ; i++)
  {
    printf(" Enter the Size (Limit value) and
    Base Address of the segment %d \n", i);
    scanf("%d %d", & SegTable[i][0] , &SegTable[i][1]);
  }
printf(" Following is the Segment Table entered
      by the user \n");
printf(" SegNo  Limit value  Base Address \n");
for(i=0; i< Numseg ; i++)
 {
  printf(" %d  %d  %d ", i, SegTable[i][0],
                  SegTable [i][1] );
  printf("\n");
 }
```

```
while(1)
{
  printf("Enter the Logical Address: (Segment NO and
          Offset value)\n");
  scanf("%d %d", &segNO, &offset);

  if(segNO > NumSeg)
  {
    printf("Wrong SegmentNo So Address binding
      not possible\n");
  }
  else if(offset > segTable[segNO][0])
  {
    printf("Offset value is greater than the limit so
      Address binding is not possible\n");
  }
  else
  {
    phyadd = segTable[segNO][1]+offset;
printf("Base Address for the given Segment Number is %d
                                        \n",
                                  segTable[segNO][1]);
printf("physical address for the given logical address
          is %d\n", phyAdd);

  }
}
return 0;
}
```

Output:

Enter the number of Segments of process1
5

Enter the Size(Limit value)and Base address for Segment0:600 219
Enter the Size(Limit value)and Base address for Segment1:14 2300
Enter the Size(Limit value)and Baseaddress for Segment2:100 90
Enter the Size(Limit value) and Baseaddress for Segment3:580 1327
Enter the Size(Limit value)and Baseaddress for Segment4:96 1952

Following is Segment table entered by the User.

SegNo	Limit value	Base Address
0	600	219
1	14	2300
2	100	90
3	580	1327
4	96	1952

Enter the Logical Address:(SegmentNo and offset value): 0 430
BaseAddress for the given Segment Number is 219
physical Address for the given Logical Address is 649

Enter the logical Address:(SegmentNo and offset value): 1 10
BaseAddress for the given Segment Number is 2300
physical Address for the given Logical Address is 2310

Enter the Logical Address:(SegmentNo and offset value): 2 500
Offset value is greater than limit so address binding not possible.

Enter the Logical Address:(SegmentNo and offset value): 3 400
BaseAddress for the given SegmentNo is 1327
physical address for the given Logical Address is 1727

Enter the Logical Address: (SegmentNo and OffsetValue): 4 112
Offset value is greater than the limit so Address binding
not possible.