**Experiment 1 : Basic Linux Commands**

**Aim :**

To study the basic commands of a linux based operating system

**Commands:**

**1) apt**

**Syntax:** [sudo] apt [option] [command] <file>

**Description:**

It is a tool for using the APT package management system. It can install,remove, upgrade, and perform other management functions on a system’s software packages.

**Options:**

**-y -** Assume the answer “yes” to any prompts

**-d -** For operation, only download files but do nothing

**--assume-no -** Assume the answer “no” to all prompts

**2) bzip2**

**Syntax:** bzip2 [option] [Filename]

**Description:**

It is used to compress and decompress files. Compared with gzip, bzip2 creates smaller archives but has a slower decompression time and higher memory use.

**Options:**

**-c -** Compress or decompress to standard output

**-d -** Force decompression

**-z -** Force compression

**3) cd**

**Syntax:** cd [option] [directory]

**Description:**

It changes the working directory. It is used to move around within the hierarchy of the file system.

**Options:**

**-L -** Follow symbolic links

**-P -** Do not follow symbolic links

**4) cat**

**Syntax:** cat [filename]

**Description:**

It reads data from files, and outputs their contents. It is the simplest way to display the contents of a file at the command line.

**Options:**

**-b -** Number non-empty input lines

**-n -** Number all input lines

**5) cal**

**Syntax:** cal

**Description:**

The command displays a simple, formatted calendar in the terminal.

**Options:**

**-h -** Don’t highlight today’s day

**-y -** year Specify a year to display

**-3 -** Display last month, this month, next month

**6) chmod**

**Syntax:** chmod [options] [permission] [filename]

**Description:**

It is used to change the permissions of files or directories.

**Options:**

**-f -** Suppress error messages

**-R -** Change files and directories recursively

**--preserve-root -**  Do not operate recursively on ‘/’

**7) chown**

**Syntax: chown [option] [filename]**

**Description:**

It changes ownership of files and directories in a Linux filesystem

**Options:**

**-L -** Traverse all symbolic links to a directory

**-P -** Do not traverse any symbolic links; operate on

the symbolic links themselves.

**8) cp**

**Syntax:** cp [option] [source] [destination]

**Description:**

It makes copies of files and directories

**Options:**

**-a -** archive When performing the copy, attempt to preserve as much of the original file structure, attributes, and associated metadata as possible

**-i -** Prompt before overwrite

**-n -** Do not overwrite an existing file

**9) date**

**Syntax:** date [option] [format]

**Description:**

It is used to print out, or change the value of, the system’s time and date information

**Options:**

**-d -** Display time described by string

**-r=FILE -** Display the last modification time of file FILE

**-R -** Output date and time in day-date-monthYear

**10) echo**

**Syntax:** echo [line of text]

**Description:**

It is used to display a line or append a line of text.

**Options:**

**-n -** Does not add newline at the end of the line of text

**-e -** Identifies and treats backlash escapes as special

**11) free**

**Syntax:** free [options]

**Descriptions:**

Displays the total amount of free and used physical and swap

memory in the system, as well as buffers used by the kernel

**Options:**

**-b -** Show the output in bytes.

**-k -** Show the output in kilobytes

**-m -** Show the output in megabytes

**-g -** Show the output in gigabytes

**12) find**

**Syntax:** find [options] filename

**Description:**

It searches for files in a directory hierarchy

**Options:**

**-P -** Never follow symbolic links

**-L -** Follow symbolic links

**-H -** Do not follow symbolic links, except while

Processing the command line arguments

**13) head**

**Syntax:** head [option] [filename]

**Description:**

It prints the first 10 lines of a file to standard output

**Options:**

**-q -** Never print headers identifying file names

**-v -** Always print headers identifying file names

**14) kill**

**Syntax:** kill [option] [pid]

**Description:**

It sends a signal to a process. If the signal is not specified, the TERM signal is sent, which terminates the process

**Options:**

**-s signal -** The name, abbreviated name, or number of the signal to be sent pid Numeric process id

**-l -** List available signal names

**-L -** List available signal names and numbers in a table

**15) ls**

**Syntax:** ls [option] [filename]

**Description:**

It lists the contents of, and optional information about, directories and files.

**Options:**

**-l -** Shows file or directory, size, modified date and time, name, owner and permission

**-r -** Display files and directories in reverse order

**-ltr -** Show latest modification date of a file or directory

**16) locate**

**Syntax:** locate [option] [name]

**Description:**

It is used to find the location of files and directories

**Option:**

**-b -** Match only the base name against patterns

**-c -** Print the number of matching entries

**-e -** Print only entries that refer to files existing

at the run time of location

**17) mkdir**

**Syntax:** mkdir [option] [directory\_name(s)]

**Description:**

It is used to create new directories. Any number of directories can be created simultaneously

**Options:**

**-m -** Used to set permission for users. It is represented using digits

**-p -** Create parent directories

**18) mv**

**Syntax:** mv [options] [source] [destination]

**Description:**

It is used to move or rename files and directories

**Option:**

**-b -** Make a backup of each existing destination file

**-f -** Overwrite existing file without prompting

**-i -** Prompt before overwriting an existing file

**-n -** Never overwrite any existing file.

**19) poweroff**

**Syntax:** poweroff [option]

**Description:**

It is used to power off the system. It invokes the shutdown tool

**Option:**

**-f -** Does not invoke shutdown and instead does what the name specifies

**-w -** Does not call shutdown or reboot system call, instead only writes the shutdown record to /var/log/wtmp

**20) pwd**

**Syntax:** pwd [option]

**Description:**

It prints the name of the working directory

**Options:**

**-P -** Print a fully resolved name for the current directory, in which all components of the name are actual directory names

**21) reboot**

**Syntax:** reboot [option]

**Description:**

It is used to reboot the system.

**Option:**

**-f -** Does not invoke shutdown and instead does what the name specifies

**-w -** Does not call shutdown or reboot system call, instead only writes the shutdown record to /var/log/wtmp

**22) scp**

**Syntax:** scp [options] [[user@]host1:]file1 [[user@]host2:]file2

**Description:**

It copies files over a secure, encrypted network connection

**Options:**

**-4 -** Forces the use of IPv4 addresses only

**-6 -** Forces the use of IPv6 addresses only

**23) split**

**Syntax:** split [option] [INPUT[PREFIX]]

**Description:**

It splits a file into pieces. The outputs are fixed-size pieces of input INPUT to files named PREFIXaa, PREFIXab, ...

**Option:**

**-a** **N -** Use suffixes of length N

**-b SIZE -**  Write SIZE bytes per output file

**24) tail**

**Syntax:** tail [option] [file]

**Description:**

It prints the last 10 lines of each file to standard output

**Options:**

**-c[+]num -** Output the last ‘num’ bytes of each file

**-f -** It will cause tail to loop forever, checking for new data at the end of the file.

**-q -** Never output headers

**-s -** num Sleep for ‘num’ seconds between checks

**25) tar**

**Syntax:** tar [option] [pathname]

**Description:**

It is used to create, modify, maintain, and extract files that are archived in the tar format

**Options:**

**-c -** Create a new tar archive

**-v -** Turn verbose mode on

**-x -** Extract a new archive

**26) time**

**Syntax:** time [option] command [arguments]

**Description:**

It reports how long a process took to execute

**Options:**

**-p -** The time will be printed in a portable POSIX format

**27) touch**

**Syntax:** touch [option] [filename]

**Description:**

It is used to change the timestamps of files. It can also be used to create empty files

**Options:**

**-a -** Set access time only

**-m -** Set modification time only

**-t -** Use numeric timestamp instead of current time

**28) wall**

**Syntax:** wall [-n] [-t timeout] [-g group] [message | file]

**Description:**

wall displays a message, or the contents of a file, or otherwise its standard input, on the terminals of all currently logged in users.

**Options:**

**-n -** No Banners

**-v -** Version information

**29) which**

**Syntax:** which [-a] filename

**Description:**

which is used to find which copy of a program is invoked when a command is executed.

**Options:**

**-a -** print all matching pathnames of each argument

**30) wget**

**Syntax:** wget [option] [URL]

**Description:**

GNU Wget is a free utility for non-interactive download of files from the Web. It supports HTTP, HTTPS, and FTP protocols, as well as retrieval through HTTP proxies.

**Options:**

**-b -** Background mode on

**-v -** Version information

**30) who**

**Syntax:** who [OPTION] [ FILE | ARG1 ARG2 ]

**Description:**

Print information about users who are currently logged in.

**Options:**

**-b -** Time of last system boot

**-d -** Print dead processes

**-q -** All users and numbers who are logged in.

**-s -** print only name, line, and time

**32) whoami**

**Syntax:** whoami [OPTION]

**Description:**

Print the user name associated with the current effective user ID. Same as id -un.

**Options:**

**-v -** Version information

**Result :**

Thus the Basic Linux Commands were executed successfully.

**Exercise 2 : BASH Programs**

**Shell programming :**

A shell is special user program which provide an interface to user to use operating system services. Shell accept human readable commands from user and convert them into something which kernel can understand. It is a command language interpreter that execute commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or start the terminal.

**Aim :**

To write Bourne Again Shell Scripts to implement the solution to the problems given.

**1. Write a shell script for Seconds to convert HH:MM:SS (24 hours format). To read the input from CLI argument and to be perform operations only using operators (+, -, \*, /, %) without using loops and control structures.**

read -p "Enter Seconds : " a

let h=$a/3600

let m=$a%3600/60

let s=$a%3600%60

echo $h:$m:$s

**Output :**

Enter Seconds : 60

0:1:0

**2. Write a shell program for Temperature Conversion [Fahrenheit to Celsius and Celsius to Fahrenheit ]. To read the input on prompting user.**

echo "1-Celcius to Farenheit"

echo "2-Farenheit to Celcius"

read n

if [ $n = 1 ]; then

read a

echo $(echo "$a\*9/5+32" | bc -l)

else

read a

let a-=32

echo $(echo "$a\*5/9" | bc -l)

fi

**Output :**

1-Celcius to Farenheit

2-Farenheit to Celcius

1

37

98.6

**3. Write a shell script for print the number of Vowels, Consonants, digits present in given string. To read the string from CLI argument.**

a=$1

echo "Vowels : "

echo $(echo $1 | grep -io [aeiou] | wc -l)

echo "Consonants : "

echo $(echo $1 | grep -io [bcdfghjklmnpqrstvwxyz] | wc -l)

echo "Digits : "

echo $(echo $1 | grep -io [0123456789] | wc -l)

**Running Command :**

bash 1.sh This12

**Output :**

Vowels : 1

Consonants : 3

Digits : 2

**4. Find the sum of digits in the given integer using shell script. To read the input on prompting user.**

let c=0

read -p "Enter number : " x

while [ $x -gt 0 ]; do

let c+=$x%10

let x/=10

done

echo $c

**Output :**

Enter number : 123

6

**5. Find whether the given two arrays are same or not.(same only if all the corresponding elements are same) using shell script.**

read -p "Enter the number of terms : " n

a=()

b=()

let i=0

while [ $i -lt $n ]; do

read x

let a[$i]=$x

let i++

done

let i=0

while [ $i -lt $n ]; do

read x

let b[$i]=$x

let i++

done

let i=0

let c=0

while [ $i -lt $n ]; do

if [ ${a[$i]} = ${b[$i]} ]; then

let c++

else

echo "Not the Same"

break

fi

let i++

done

if [ $c = $n ]; then

echo "They are the same"

fi

**Output :**

Enter the number of terms : 5

1 2 3 4 5

1 2 3 4 5

They are the same

**6. To write a shell program on create an array of binary equivalents for the given array of integers.**

function bin(){

let n=$1

b=""

while [ $n -gt 0 ]; do

b="$((n%2))$b"

let n=$n/2

done

echo $b

}

read n

a=()

let i=0

while [ $i -lt $n ]; do

read a[$i]

let i++

done

let i=0

while [ $i -lt $n ]; do

echo $(bin ${a[$i]})

let i++

done

**Output :**

1 2 3 4 5

1 10 11 100 101

**7. The students of a class are sitting in M rows with each row consisting of N seats. The students should sit in roll number order from row 1(occupying first N students) to row M. If a student is absent then the seat is kept as vacant. Now you are given a binary matrix of order MxN, where 1 denotes the seat is occupied by the student for the seat and 0 denotes that student is absent. Print the roll numbers of the absentees as per the given binary matrix.**

read n

read m

a="Absentees : "

echo "Enter the binary array : \n"

let i=0

while [ $i -lt $m ]; do

let j=0

while [ $j -lt $n ]; do

read x

if [ $x == 0 ]; then

a="$a $(($i\*$n+$j+1))"

fi

let j++

done

let i++

done

echo $a

**Output :**

2

2

1 1 1 0

Absentees : 4

**8. To write a shell program on check whether the given integer and its reverse are same or not, using function.**

let c=0

read -p "Enter number : " x

let t=x

while [ $x -gt 0 ]; do

let c\*=10

let c+=$x%10

let x/=10

done

if [ $c == $t ] ; then

echo "Palindrome"

else

echo "Not Palindrome"

fi

**Output :**

Enter number : 112211

Palindrome

**9. To a shell program using simple calculator function – Read an integer followed by an operator. If the operator is + or – or \* or /, then read the next operand and compute the result. If the operator is =, then print the result so far accumulated. Perform integer type operations.**

let r=0

while true; do

read -p "Enter a number : " x

read -p "Enter an operand : " o

if [ $o == "=" ]; then

echo $r

break

elif [ $o == '+' ]; then

read -p "Enter a number : " y

let r=$x+$y

elif [ $o == '-' ]; then

read -p "Enter a number : " y

let r=$x-$y

elif [ $o == 'x' ]; then

read -p "Enter a number : " y

let r=$x\*$y

elif [ $o == '/' ]; then

read -p "Enter a number : " y

let r=$x/$y

fi

Done

**Output :**

Enter a number : 5

Enter an operand : +

Enter a number : 5

Enter a number : 5

Enter an operand : =

10

**10.To write a shell program using function to find whether the given string is palindrome or not. Create an array of strings in the main function and find how many strings in the string array are palindrome.**

read -p “Enter the string : ” a

s = ${a[10]}

r = “”

l = ${#string}

for((i=($l-1); i>=0; i--)) do

reverse = “$reverse${string[$i]}”

Done

if [ $a == $s ]; then

echo “Palindrome”

else

echo “Not Palindrome”

fi

**Output :**

**11.To write a shell script using recursive function to find the nth Fibonacci term in the Fibonacci series. Using the recursive function, generate the first m terms in Fibonacci series.**

function fib(){

let n=$1

if [ $n -le 1 ]

then

echo 1

else

let x=$(fib $[$n-1])

let y=$(fib $[$n-2])

let x=$x+$y

echo $x

fi

}

let i=0

while [ $i -le $1 ]; do

echo $(fib $i)

let i++

done

**Output :**

1 1 2 3 5 8 13

**Result :**

Thus the given programs were successfully implemented using the Bourne Again Shell Scripts.

**Experiment 3 : Grep, Sed and AWK**

**Aim :**

To use the linux based commands grep, sed and awk to solve various problems on text searching, replacement and manipulation.

**1. Study the following Linux commands : sed, grep and awk (Command name,**

**Syntax, Description : , options)**

**grep -** Global Regular Expression Print

**Syntax :** grep [options] <string> <files>

**Description :**

Grep is used to search for a string or a pattern in a file or a stream and then return the lines that match.

**Options :**

**-e :** extendedregex

**-i :** case insensitive

**-c :** line count

**sed -** Search and edit string

**Syntax :** sed [options] <script> <file>

**Description :**

sed (stream editor) is a Unix utility that parses and transforms text, using a simple, compact programming language.sed was based on the scripting features of the interactive editor ed

**Options :**

**-n :** Turns on the silent mode

**-p :** Print the matches

**AWK -** Aho Weinberger Kernighan Text Editor

**Syntax :** awk [options] <file>

**Description :**

AWK is a programming language designed for text processing and typically used as a data extraction and reporting tool. It is a standard feature of most Unix-like operating systems.

**Options :**

**-f :** file to be used

**-v :** Variable declaration

**2. Write the grep commands for each of the following task.**

**a. Find all subdirectories within a directory.**

ls -F grep ‘/’

**b. Write a regex that matches the emails of the form** [**userid@domain.edu**](mailto:userid@domain.edu)**.Where userid is one of more word characters or ‘+’ and the domain is one or more word characters.**

grep “\b[A-Za-z0-9]\*.@[A-Za-z0-9].edu”

**c. Find all patterns that begins with “r” or any digit from “0-9”.**

grep “\b[r0-9]\*.\b”

**d. Use the sed command to delete the first character and last character in each line of a file.**

sed -e “s:\(.\)::” -e “s:\($\)::”

**e. Use the grep command to find how many lines of a file contain a given**

**word. The filename and the word are provided as inputs.**

grep word filename | wc -l

**3. Write a sed command that swaps the first and second words in each line in a file.**

sed 's:\([^ ]\) \([^ ]\):\2 \1:' file

**4. Use the awk command**

**a. Print the lines which matches with the pattern.**

read -p "Enter the pattern : " a

awk '/$a/ {print}'

**b. Print number of employees in Technology department**

echo Number of people in Technology :

awk '/Technology/ {n++} END{print n}' employee.txt

**c. Print the number of occurrences of each word in its input file.**

**5. Write a shell script that takes a command line argument and reports on whether it is directory, a file, or something else.**

if [ -f $1 ]; then

echo "file"

elif [ -d $1 ]; then

echo "Directory"

else

echo "Something else"

**Output :**

bash 1.sh a.txt

file

**6. Write a shell script that accepts one or more file name as arguments and converts all of them to uppercase, provided they exist in the current directory.**

read -p "Enter the filename : " a

awk '{print toupper($0)}' $a > tmp

cat tmp >a

**Output :**

Enter the filename : a.txt

THIS IS JUST SOME SAMPLE TEXT.

**7. Write a shell script program and C program to display “HELLO WORLD” Compare the running time of both the programs using time command.**

**Shell Script**

echo Hello World

**C**

#include <stdio.h>

int main(){

printf(“Hello World”);

}

**8. Write a shell script that determines the period for which a specified user is working on the system.**

read -p "Enter the username : " u

finger $u | grep 'On since' | sed 's:\([^ ]\*\) \([^ ]\* [^ ]\* [^ ]\* [^ ]\* [^ ]\* [^ ]\* [^ ]\*\) \(.\*\):\2:'

**9. Write a shell script that accepts a filename, starting and ending line numbers**

**as arguments and displays all the lines between the given line numbers.**

read -p "Enter the filename, start line and end line : " a b c

tail $a -n $b | head -n $(($c-$b))

**Output :**

Enter the filename, start line and end line : a.txt 10 15

This

Is just

Some

Sample

Text

**10.Write a shell script that deletes all lines containing a specified word in one or**

**more files supplied as arguments to it.**

read -p "Enter the file and the word to search for : " a b

cat $a

sed -i '/$b/D' $a

cat $a

**Output :**

Enter the file and the word to search for : a.txt Del

A

B

Del

C

Del

A

B

C

**11.Write a shell script to perform the following string operations:**

**a. To extract a substring from a given string**

read -p "Enter string, start and end points : " a b c

echo ${a:$b:$(($c-$b))}

**Output :**

Enter string, start and end points : Refrigerator 1 3

Ref

**b. To find the length of a given string**

read -p "Enter string to find length : "

echo $s | wc -c

**Output :**

Enter string to find length : Confidence

11

**12.Create a file consisting of structures with regno, name and GPA. Write shell program to find the students with GPA above the given floating point value.**

read -p "Enter GPA value : " a

awk '{if($3>$a) print}' student\_12.txt

**Output :**

Enter GPA value : 8

2017503501 Ajay 9.0

**13.Create a file consisting of the students of a class. Separate them into two different files so that males and females are in separate files using shell script.**

echo "After running this male student will be found in m.txt and female students will be in f.txt..."

awk '{if($4=="M") print}' student\_13.txt > m.txt

awk '{if($4=="F") print}' student\_13.txt > f.txt

**Output :**

After running this male student will be found in m.txt and female students will be in f.txt...

**14.A file contains information about the employees of an organization. The information includes employee number, salary, and age. Write a shell program to count the number of records in that file and also print the employee numbers for those who getting salary less than Rs.4500 and aged more than 35.**

awk 'BEGIN{print "Employee num"}{if($2 < 4500 && $3 > 35) {n++; print $1;}} END{print "Number of employees : "n}' employee\_14.txt

**Output :**

**Employee num**

21

20

Number of Employees : 2

**15. Write the shell script to do the following:**

**Input:**

**a. There are twelve master files having the name of the three letters of the months of year. These master file contain two fields: item name and accepted price**

**b. There is TRANS file which contains three fields: month, item name and a quantity. There can be any number of records for each item.**

echo "" >tmp15.txt

while read l; do

i=$(echo $l | awk '{print $1}')

m=$(echo $l | awk '{print $2}')

q=$(echo $l | awk '{print $3}')

let v=$(grep $i $m.txt | awk '{print $2}')

let t=v\*q

if [ $(grep $i tmp15.txt | wc -l) = 0 ]; then

echo "$i $t">>tmp15.txt

else

let x=$(grep $i tmp15.txt | awk '{print $2}')

let x=$x+$t

grep -v $i tmp15.txt >tmp152.txt

cat tmp152.txt>tmp15.txt

echo "$i $x">>tmp15.txt

fi

done < TRANS.txt

cat tmp15.txt

**Result :**

Thus grep, sed and awk were successfully used to solve the given problems on text searching, replacement and manipulation.

**Exercise 4 : System Calls**

**System Calls :**

In computing, a system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.

Types of System Calls :

Process management

File management

Device management

Information management

Communication

**open() :**

The open() system call opens the file specified by pathname. If the specified file does not exist, it may optionally (if O\_CREAT is specified in flags) be created by open().

**Syntax :** int open(const char \*pathname, int flags, mode\_t mode);

**read() :**

read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf. On files that support seeking, the read operation commences at the current file offset, and the file offset is incremented by the number of bytes read. If the current file offset is at or past the end of file, no bytes are read, and read() returns zero.

**Syntax :** ssize\_t read(int fd, void \*buf, size\_t count);

**write() :**

write() writes up to count bytes from the buffer pointed buf to the file referred to by the file descriptor fd. The number of bytes written may be less than count if, for example, there is

insufficient space on the underlying physical medium, or the RLIMIT\_FSIZE resource limit is encountered, or the call was interrupted by a signal handler after having written less than count bytes.

**Syntax :** ssize\_t write(int fd, const void \*buf, size\_t count);

**Aim :**

To use the various system calls available in the Linux operating system.

**1. To write a C program to count number of lines in file using system calls.**

#include<stdio.h>  
#include<fcntl.h>  
#include<unistd.h>  
int main(){  
 char a[2];  
 int fd=open("a.txt", O\_RDONLY), c=-1;  
 int x=1;  
 while(x){  
 x = read(fd, a, 1);  
 if(a[0]=='\n')  
 c++;  
 }  
 printf("%d",c);  
}

**Output :**

10

**2. To implement in C the following UNIX commands using System calls :**

**i. cat**

#include<stdio.h>

#include<fcntl.h>

#include<unistd.h>

#include<string.h>

int main(){

char a[100], b[100];

int x, s, d, f=1;

printf("Enter input file or stdin : ");

scanf("%s",a);

printf("Enter output file or stdout : ");

scanf("%s",b);

printf("Do you want to append (0/1) : ");

scanf("%d",&x);

if(strcmp(a, "stdin"))

s = open(a, O\_RDONLY);

else

s = 0;

if(strcmp(b, "stdout"))

d = open(b, O\_WRONLY);

else

d = 1;

if(x)

lseek(d, 0, SEEK\_END);

while(f){

char c[2];

f = read(s, c, 1);

printf("%c",\*c);

write(d, c, 1);

}

}

**Output :**

Enter input file or stdin : stdin

Enter output file or stdout : stdout

Do you want to append (0/1) : 0

THis is some sample text.

THis is some sample text.

^Z

**ii. cp**

#include<stdio.h>

#include<fcntl.h>

#include<unistd.h>

int main(){

char a[2], s[1000], d[1000];

scanf("%s %s",s,d);

int src=open(s, O\_RDONLY), dest=open(d, O\_CREAT|O\_WRONLY, 999);

int x=1;

while(x){

x = read(src, a, 1);

write(dest, a, 1);

}

}

**Output :**

a.txt b.txt

**iii. mv**

#include<stdio.h>

#include<fcntl.h>

#include<unistd.h>

int main(){

char a[2], s[1000], d[1000];

scanf("%s %s",s,d);

int src=open(s, O\_RDONLY), dest=open(d, O\_CREAT|O\_WRONLY, 999);

int x=1;

while(x){

x = read(src, a, 1);

write(dest, a, 1);

}

unlink(s);

}

**Output :**

a.txt b.txt

**3. Determine the size of a file using the lseek command. Once you found out the size, calculate the number of blocks assigned for the file. Compare these results with the similar results obtained when using the function stat.**

#include<stdio.h>

#include<fcntl.h>

#include<unistd.h>

#include<math.h>

int main(){

char a[2], s[1000], d[1000];

scanf("%s",s);

int src=open(s, O\_RDONLY);

long int x=lseek(src, 0, SEEK\_END);

printf("Size : %ld\n", x);

double y = ceil(x/4096.0);

printf("Blocks : %.0lf", y);

}

**Output :**

A.txt

Size : 10

Blocks : 10

**Result :**

Thus the various system call of the c language provided by the Linux operating system were used to interface with the filesystem and the given problems were solved.

**Exercise 5 : Process Management**

**fork() :**

The fork() system call is used to create processes. When a process (a program in execution) makes a fork() call, an exact copy of the process is created. Now there are two processes, one being the parent process and the other being the child process. The process which called the fork() call is the parent process and the process which is created newly is called the child process.

**Syntax :** pid\_t fork(void);

**exec() :**

The exec() system call is also used to create processes. But there is one big difference between fork() and exec() calls. The fork() call creates a new process while preserving the parent process. But, an exec() call replaces the address space, text segment, data segment etc. of the current process with the new process.

**Syntax :** execvp(char \* file, char \* argv[])

**Pstree :**

The pstree command displays a tree of processes.

**Syntax :** pstree [options]

pstree shows running processes as a tree. The tree is rooted at either pid or init if pid is omitted. If a user name is specified, all process trees rooted at processes owned by that user are shown.

**Ps :**

ps command Linux provides us a utility called ps for viewing information related with the processes on a system which stands as abbreviation for “Process Status”. ps command is used to list the currently running processes and their PIDs along with some other information depends on different options. It reads the process information from the virtual files in /proc file-system. /proc contains virtual files, this is the reason it’s referred as a virtual file system.

ps provides numerous options for manipulating the output according to our need.

**Syntax :** ps [options]

**Aim :**

To implement the generation and usage of child processes to solve the given problems optimally.

**1. Write a program in C that creates a child process, waits for the termination of the child and lists its PID, together with the state in which the process was terminated (in decimal and hexadecimal).**

#include<stdio.h>

int main(){

int pid=10, status;

if((pid=fork())==0)

printf("Child Process(PID=%d)...Terminating",getpid());

else{

waitpid(pid, &status, 0);

printf("\n\nProcess ID of the child : %d\nExit Status of the child : %d\n\n", pid, status);

}

}

**Output :**

Child Process(PID=3488)...Terminating

Process ID of the child : 3489

Exit Status of the child : 0

**2. To write a C program for creation of orphan process and zombie process.**

**Orphan**

#include<stdio.h>

int main(){

int a=1, pid=10, status;

if((pid=fork())==0){

sleep(10);

printf("Child Process(PPID=%d)...Terminating",getppid());

}

else{

printf("Exiting Parent(PID=%d).",getpid());

}

}

**Output :**

Exiting Parent(PID=3488)

Child Process(PPID=3489)...Terminating

**Zombie**

#include<stdio.h>

int main(){

int a=1, pid=10, status;

if((pid=fork())==0){

printf("Child Process(PPID=%d)...Terminating",getppid());

}

else{

sleep(10);

printf("Exiting Parent(PID=%d).",getpid());

}

}

**Output :**

Child Process(PPID=3489)...Terminating

Exiting Parent(PID=3488).

**3. In a C program, print the address of the variable and enter into a long loop.**

**a) Start three to four processes of the same program and observe the printed address values.**

#include<stdio.h>

int main(){

fork();

fork();

fork();

fork();

int a;

printf("%d\n",&a);

//while(1) a=1;

}

**b) Show how two processes which are members of the relationship parent- child are concurrent from execution point of view, initially the child is copy of the parent, but every process has its own data.**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <unistd.h>

int main(){

int pid;

int n=0;

int status;

pid=fork();

while(n <10){

if(pid==0){

printf(“Child : %d\n”,n);

n=n+2;

sleep(1);

}

else{

printf(“Parent : %d\n”,n);

n=n+3;

sleep(1);

}

}

}

**Output :**

Child : 0

Parent : 2

**4. Test the source code below:**

**for(i = 1; i ≤ 10; i + +)**

**{**

**fork();**

**printf(“The process with the PID=%d”,getpid());**

**}**

**Output :**

10240 Processes get created.

**5. Write two programs : one called client.c, the other called server.c. The client program lists a prompter and reads from the keyboard two integers and one of the characters ’+’ or ’-’. The read information is transmitted with the help of the system call execl to a child process, which executes the server code. After the child (server) process finishes the operation, it transmits the result to parent process (client) with the help of the system call exit. The client process prints the result on the screen and also reprints the prompter, ready for a new reading.**

Server:

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <unistd.h>

int main( int argc, char \*\*argv)

{

int a;

int b;

int c=0;

a = \*argv[0] – 48;

b = \*argv[1] – 48;

printf(“\n %d %d “, a, b );

if(\*argv[2] == ‘+’)

c=a+b;

else

c=a-b;

return res;

}

Client:

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main()

{

int pid;

char op1, op2;

char sign;

int status;

printf(“>”);

scanf(“%c %c %c”, &op1, &sign, &op2);

pid = fork();

if (pid == 0)

{

if (execl(“./child.exe”, &op1, &op2, &sign, NULL)<0)

perror(“eroare”);

}

else if (pid>0)

{

waitpid(pid, &status, 0);

printf(“result %d\n”, WEXITSTATUS(status));

}

exit(0);

}

**Output :**

8 3 -

5

**6. Write a C program that takes a file name as a command line parameter and sorts a set of integers stored in the file. Now write a C program (xsort.c) that implements a command called "xsort" that you will invoke from the shell prompt. The syntax of the command is "xsort <filename>". When you type the command, the command opens a new xterm window, and then sorts the integers stored in the file <filename> using the program "sort1". Look up the man pages for xterm, fork and the different variations of exec\* calls (such as execv, execve, execlp etc.) to do this problem.**

#include <stdio.h>

void sort(int \*a, int n) {

int i, j;

for (i = 0; i < n-1; i++)

for (j = 0; j < n-i-1; j++)

if (a[j] > a[j+1]) {

int t = a[j];

a[j]=a[j+1];

a[j+1]=t;

}

}

void main (int argc,char\* argv[]) {

char\* s=argv[1];

FILE\* fd= fopen (a.txt, "r");

int s=0, n=0, a[1000];

fscanf(file,"%d", &s);

while(!feof (file)) {

a[n]=s;

n++;

fscanf (file,"%d", &s);

}

sort(a, n);

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

printf("%d\n", a[i]);

}

**Result :**

Thus the given problems were solved using the creation and usage of child processes.

**Exercise 6 : Process with PIPES**

**pipe() :**

Conceptually, a pipe is a connection between two processes, such that the standard output from one process becomes the standard input of the other process.

**Syntax :** int pipe(int fds[2]);

**Aim :**

To use PIPEs as the inter process communication mechanism to communicate between the various processes.

**1. Copy a file using pipes to communicate between the parent and the child processes.**

#include <stdio.h>

#include <fcntl.h>

int main(int arg\_count, char \*arg\_pointer[]){

int p1[2], p2[2], f1, f2;

char a[100];

pipe(p1);

pipe(p2);

f1 = open(arg\_pointer[1], O\_RDONLY);

f2 = open(arg\_pointer[2], O\_WRONLY|O\_CREAT);

if(fork()){

while(1){

int x = read(p1[0], a, 100);

int y = write(f2, a, x);

if(y<0)

write(p2[1], "-1", 2);

else

write(p2[1], "0", 1);

if(x<100){

printf("Copy Successful.");

close(p1[0]);

break;

}

}

}

else{

while(1){

int x = read(f1, a, 100);

//printf("%s",a);

write(p1[1], a, x);

if(x<100){

//printf("%d",x);

close(p1[1]);

break;

}

char c[2];

if(read(p2[0], c, 2)==2){

perror("Some Error Occured...\n\n");

return -1;

}

}

}

}

**2. Write a C program that will search an array of integers for another given integer and the search is done in parallel by two child processes.**

#include<stdio.h>

#include<stdlib.h>

int main(){

int n, x, i;

scanf("%d %d",&n,&x);

int \*a = (int\*) malloc(n\*sizeof(int));

int p1[2], p2[2];

char ax[1000];

pipe(p1);

pipe(p2);

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

scanf("%d", &a[i]);

if(fork()){

char f=1;

int i;

for(i=0; i<=n/2; i++)

if(a[i]==x){

if(i==0){

write(p1[1], "0 ", 2);

}

f=0;

char s[10];

int t=i, j=0;

while(t>0){

s[j++]=t%10+'0';

t/=10;

}

s[j]=' ';

s[j+1]=0;

//printf("%s",s);

write(p1[1], s, j);

}

if(f)

write(p1[1], "-1", 2);

write(p1[1], "\0", 1);

close(p1[1]);

}

else if(fork()){

char f=1;

int i;

for(i=n/2+1; i<n; i++)

if(a[i]==x){

f=0;

char s[10];

int t=i, j=0;

while(t>0){

s[j++]=t%10+'0';

t/=10;

}

s[j]=' ';

s[j+1]=0;

write(p2[1], s, j);

//printf("%d",i);

}

if(f)

write(p2[1], "-1", 2);

write(p2[1], "\0", 1);

close(p2[1]);

}

else{

wait(NULL);

read(p1[0], ax, 1000);

printf("%s ", ax);

read(p2[0], ax, 1000);

printf("%s ",ax);

}

}

**Output :**

5 1

1 2 3 4 5

1 -1

**3. You will write two simple programs pipe reader.c and pipe writer.c that use a named pipe to communicate. The pipe reader program will set up a named pipe using mkfifo(), open it read only, and read strings from it until it receives the string exit. The writer will open the named pipe file, read strings from the user and write them to the named pipe. When the user enters exit, the program will write the string to the pipe and then exit.**

**Reader**

#include<stdio.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <string.h>

int main(){

char a[1000];

mkfifo("the\_wire", 0777);

int fd = open("the\_wire", O\_RDWR), f=1;

while(f){

printf("Waiting for input...");

read(fd, a, 1000);

printf("Got it : '");

printf("%s",a);

printf("'\n");

if(strcmp(a, "exit")==0)

f=0;

}

close(fd);

puts("Exiting...");

}

**Writer**

#include<stdio.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <string.h>

int main(){

char a[1000];

mkfifo("the\_wire", 0777);

int fd = open("the\_wire", O\_RDWR), f=1;

while(f){

printf("Enter the input : ");

scanf("%s", a);

a[strlen(a)+1]=0;

write(fd, a, strlen(a)+1);

printf("Writing to buffer...done\n");

if(strcmp(a, "exit")==0)

f=0;

}

close(fd);

puts("Exiting...");

}

**Output :**

**Reader :**

Creating named pipe: /tmp/mypipe

Waiting for input...Got it: ’Oh! God’

Waiting for input...Got it: ’OS lab trouble’

Waiting for input...Got it: ’exit’

Exiting

**Writer :**

Opening named pipe: /tmp/mypipe

Enter Input: Oh! God

Writing buffer to pipe...done

Enter Input: OS lab trouble

Writing buffer to pipe...done

Enter Input: exit

Writing buffer to pipe...done

Exiting

**4. A C program is to be written to print all numbers between 1 and 1000 (inclusive) that are not (evenly) divisible by either 2 or 3. This problem is to be solved using three processes (P0, P1, P2) and two one-integer buffers (B0 and B1).**

**Result :**

Thus the PIPEs were used successfully to communicate between the various processes to solve the given problems.

**Exercise 7 : Process Synchronization**

**Aim :**

To use the various mechanisms of process synchronization to solve the given problems.

**1. Solve the Producer & consumer problem with the number of producers as 1 and the number of consumers as 2.**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

pthread\_t \*pr, \*con;

sem\_t mutex, empty, full;

int \*buffer, buffer\_pos=-1, prod\_count, con\_count, buffer\_len;

int produce(pthread\_t self)

{

int i = 0;

int p = 1 + rand()%40;

while(!pthread\_equal(\*(pr+i),self) && i < prod\_count)

i++;

printf("Producer %d produced %d \n",i+1,p);

return p;

}

void consume(int p,pthread\_t self){

int i = 0;

while(!pthread\_equal(\*(con+i),self) && i < con\_count)

i++;

printf("Buffer:");

for(i=0;i<=buffer\_pos;++i)

printf("%d ",\*(buffer+i));

printf("\nConsumer %d consumed %d \nCurrent buffer len: %d\n",i+1,p,buffer\_pos);

}

void\* producer(void \*args)

{

while(1)

{

int p = produce(pthread\_self());

sem\_wait(&empty);

sem\_wait(&mutex);

++buffer\_pos;

\*(buffer + buffer\_pos) = p;

sem\_post(&mutex);

sem\_post(&full);

sleep(1 + rand()%3);

}

return NULL;

}

void\* consumer(void \*args)

{

int c;

while(1)

{

sem\_wait(&full);

sem\_wait(&mutex);

c = \*(buffer+buffer\_pos);

consume(c,pthread\_self());

--buffer\_pos;

sem\_post(&mutex);

sem\_post(&empty);

sleep(1+rand()%5);

}

return NULL;

}

int main(void)

{

int i,err;

srand(time(NULL));

sem\_init(&mutex,0,1);

sem\_init(&full,0,0);

printf("Enter the number of pr:");

scanf("%d",&prod\_count);

pr = (pthread\_t\*) malloc(prod\_count\*sizeof(pthread\_t));

printf("Enter the number of con:");

scanf("%d",&con\_count);

con = (pthread\_t\*) malloc(con\_count\*sizeof(pthread\_t));

printf("Enter buffer capacity:");

scanf("%d",&buffer\_len);

buffer = (int\*) malloc(buffer\_len\*sizeof(int));

sem\_init(&empty,0,buffer\_len);

for(i=0;i<prod\_count;i++){

err = pthread\_create(pr+i,NULL,&producer,NULL);

if(err != 0){

printf("Error creating producer %d: %s\n",i+1,strerror(err));

}else{

printf("Successfully created producer %d\n",i+1);

}

}

for(i=0;i<con\_count;i++){

err = pthread\_create(con+i,NULL,&consumer,NULL);

if(err != 0){

printf("Error creating consumer %d: %s\n",i+1,strerror(err));

}else{

printf("Successfully created consumer %d\n",i+1);

}

}

for(i=0;i<prod\_count;i++)

{

pthread\_join(\*(pr+i),NULL);

}

for(i=0;i<con\_count;i++)

{

pthread\_join(\*(con+i),NULL);

}

return 0;

}

**Output :**

0 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

1 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

41 42 43 44 45 46 47 48 49 50

sum in parent is = 1254

**2. Solve the Dining Philosophers’ problem in the following two methods :**

**i) Each philosopher attempts to grab both the forks (left and right) simultaneously. If it can, it eats. If not, it waits.**

**ii) Each philosopher grabs the two forks one by one – first the left fork, and then after some waiting the right fork. This version may lead to deadlocks. The parent process checks at regular intervals whether a deadlock has occurred. If so, it chooses a philosopher randomly and releases the fork (the left one actually) grabbed by him.**

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N

int state[N];

int phil[N] = { 0, 1, 2, 3, 4 };

sem\_t mutex;

sem\_t S[N];

void test(int phnum)

{

if (state[phnum] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)

{

state[phnum] = EATING;

sleep(2);

printf("Philosopher %d takes fork %d and %d\n", phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is Eating\n", phnum + 1);

sem\_post(&S[phnum]);

}

}

void take\_fork(int phnum)

{

sem\_wait(&mutex);

state[phnum] = HUNGRY;

printf("Philosopher %d is Hungry\n", phnum + 1);

test(phnum);

sem\_post(&mutex);

sem\_wait(&S[phnum]);

sleep(1);

}

void put\_fork(int phnum)

{

sem\_wait(&mutex);

state[phnum] = THINKING;

printf("Philosopher %d putting fork %d and %d down\n", phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is thinking\n", phnum + 1);

test(LEFT);

test(RIGHT);

sem\_post(&mutex);

}

void\* philosopher(void\* num)

{

while (1)

{

int\* i = num;

sleep(1);

take\_fork(\*i);

sleep(0);

put\_fork(\*i);

}

}

int main()

{

int i;

pthread\_t thread\_id[N];

sem\_init(&mutex, 0, 1);

for (i = 0; i < N; i++)

sem\_init(&S[i], 0, 0);

for (i = 0; i < N; i++)

{

pthread\_create(&thread\_id[i], NULL, philosopher, &phil[i]);

printf("Philosopher %d is thinking\n", i + 1);

}

for (i = 0; i < N; i++)

pthread\_join(thread\_id[i], NULL);

}

**Output :**

Philosopher 1 is thinking

Philosopher 2 is thinking

Philosopher 3 is thinking

Philosopher 4 is thinking

Philosopher 5 is thinking

Philosopher 1 is Hungry

Philosopher 3 is Hungry

Philosopher 2 is Hungry

Philosopher 2 is Eating

Philosopher 2 takes fork 1 and fork 2

Philosopher 4 is Hungry

Philosopher 5 is Hungry

Philosopher 5 is Eating

Philosopher 5 takes fork 4 and fork 5

Philosopher 2 put down the fork 1 and 2

Philosopher 2 is THINKING

**3. Solve the Reader-Writer problem with the readers as the first priority.**

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

sem\_t mutex,writeblock;

int data=0,rcnt=0;

void \*reader(void \*arg){

int f;

f=((int)arg);

sem\_wait(&mutex);

rcnt=rcnt+1;

if(rcnt==1)

sem\_wait(&writeblock);

sem\_post(&mutex);

printf("\ndata read by th reader %d is %d\n",f,data);

sleep(1);

sem\_wait(&mutex);

rcnt=rcnt-1;

if(rcnt==0)

sem\_post(&writeblock);

sem\_post(&mutex);

}

void \*writer(void \*arg){

int f;

f=((int) arg);

sem\_wait(&writeblock);

data++;

printf("\ndata written by the writer %d is %d\n",f,data);

sleep(1);

sem\_post(&writeblock);

}

int main(){

int i,b;

pthread\_t rtid[5],wtid[5];

sem\_init(&mutex,0,1);

sem\_init(&writeblock,0,1);

for(i=0;i<=2;i++){

pthread\_create(&wtid[i],NULL,writer,(void \*)i);

pthread\_create(&rtid[i],NULL,reader,(void \*)i);

}

for(i=0;i<=2;i++){

pthread\_join(wtid[i],NULL);

pthread\_join(rtid[i],NULL);

}

return 0;

}

**Output :**

Child 1 access

Child 2 access

pid= 11922, var = 0

pid= 11921, var = 0

parent access

pid= 11920, var = 1

Child 2 access

Child 1 access

pid= 11922, var = 1

pid= 11921, var = 1

Child 1 access

Child 2 access

pid= 11921, var = 1

pid= 11922, var = 1

parent access

pid= 11920, var = 2

**Result :**

Thus the various process synchronization mechanisms were successfully used and the given problems were successfully solved.

**Exercise 8 : Process unrelated Shared Memory**

**Aim :**

To use the shared memory concept as the mechanism of inter process communication to solve the given problems.

**1. Parallel Merge Sort**

#include<stdio.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <stdlib.h>

void sort(int \*a, int l, int r, int min){

if(l-r<=min){

int i, j, n=r-l;

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

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

if(a[i]<a[j]){

int t = a[i];

a[i] = a[j];

a[j] = t;

}

}

else{

int m = (l+r)/2;

int pl = fork();

if(pl==0)

sort(a, l, m, min);

else{

int pr = fork();

if(pr==0)

sort(a, m+1, r, min);

else{

wait(NULL);

int L[m-l], R[r-m-1], i;

for(i=l; i<m; i++)

L[i-l] = a[i];

for(i=m+1; i<r; i++)

R[i-m-1] = a[i];

int x=0, y=0;

while(x<m-l&&y<r-m-1){

if(L[x]<R[y])

a[x+y] = L[x++];

else

a[x+y] = L[y++];

}

while(x<m-l)

a[x+y] = L[x++];

while(y<r-m-1)

a[x+y] = R[y++];

}

}

}

}

int main(){

int fd = shm\_open("arr", O\_CREAT|O\_RDWR, 0777), len = 100, i;

ftruncate(fd, 5000);

int \*a = mmap(0, 5000, PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, 0);

for(i=0; i<10; i++)

a[i]=rand()%25;

puts("\n");

sort(a, 0, 10, 2);

for(i=0; i<10; i++)

printf("%d ",a[i]);

}

**Output :**

6 7 8 12 12 15 16 23 24 25

**2. Communication using fifo**

**Reader**

#include<stdio.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <string.h>

int main(){

char a[1000];

mkfifo("the\_wire", 0777);

int fd = open("the\_wire", O\_RDWR), f=1;

while(f){

printf("Waiting for input...");

read(fd, a, 1000);

printf("Got it : '");

printf("%s",a);

printf("'\n");

if(strcmp(a, "exit")==0)

f=0;

}

close(fd);

puts("Exiting...");

}

**Writer**

#include<stdio.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <string.h>

int main(){

char a[1000];

mkfifo("the\_wire", 0777);

int fd = open("the\_wire", O\_RDWR), f=1;

while(f){

printf("Enter the input : ");

scanf("%s", a);

a[strlen(a)+1]=0;

write(fd, a, strlen(a)+1);

printf("Writing to buffer...done\n");

if(strcmp(a, "exit")==0)

f=0;

}

close(fd);

puts("Exiting...");

}

**Output :**

**Reader :**

Creating named pipe: /tmp/mypipe

Waiting for input...Got it: ’Oh! God’

Waiting for input...Got it: ’OS lab trouble’

Waiting for input...Got it: ’exit’

Exiting

**Writer :**

Opening named pipe: /tmp/mypipe

Enter Input: Oh! God

Writing buffer to pipe...done

Enter Input: OS lab trouble

Writing buffer to pipe...done

Enter Input: exit

Writing buffer to pipe...done

Exiting

**Result :**

Thus the shared memory concept was successfully used as the inter process communication mechanism to solve the given problems.