

## 1. Learn the use of basic UNIX commands:

### a. To access information using : date, history, man, who, whoami, uptime, finger, cal:

#### >date :

Date command is used to display the system date and time

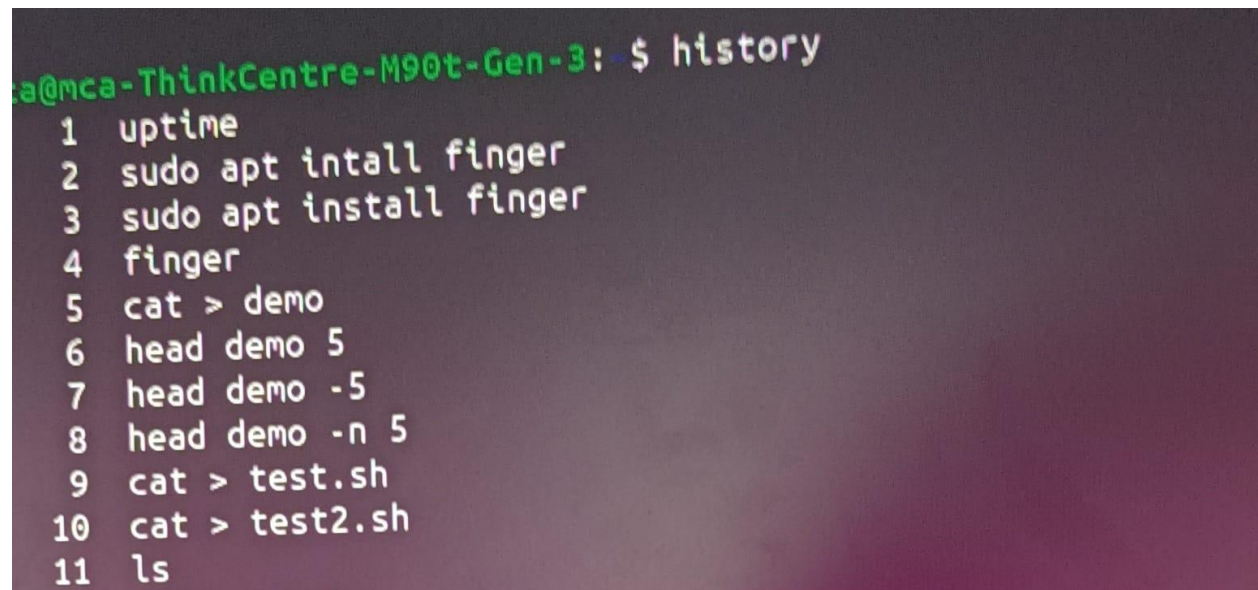
It is also used to set date and time of the system.

By default, 'date' command displays the date in the time zone on which unix operating system is configured.

```
$ date
o/p:
Tue Apr 1 02:32:44 IST 2024
```

#### >history:

**History** command is used to give the previously executed commands.



```
a@mca-ThinkCentre-M90t-Gen-3:~$ history
1  uptime
2  sudo apt install finger
3  sudo apt install finger
4  finger
5  cat > demo
6  head demo 5
7  head demo -5
8  head demo -n 5
9  cat > test.sh
10 cat > test2.sh
11 ls
```

#### >man:

Man command is used to display the user manual of any command that we can run on the terminal.

It provides a detailed view of the command which includes: name, description, return values, errors, examples, etc.

```
$man ls
```

LS(1)

User Commands

LS(1)

## NAME

ls - list directory contents

## SYNOPSIS

ls [OPTION]... [FILE]...

## DESCRIPTION

List information about the FILES (the current directory by default).

Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.

### >who:

Who command is used to get information about currently logged users on to the system.

- This command is used to display the following information for each user currently logged into the system.
- Login name of the users.
- Terminal line numbers
- Login time of the users into the system
- Remote hostname of the users.

```
$ who
```

```
exam1 pts/0 1980-04-01 02:26 (192.168.1.248)
```

>whoami: command is used to display the username of the current system.

```
$ who am i
```

```
exam1 pts/0 1980-04-01 02:26 (192.168.1.248)
```

### >uptime:

Command is used to find out how long the system is active. This command returns set of values which contains: current time, amount of time the system in running state, number of user currently logged into and the load time.

```
$ uptime
```

```
03:02:45 up 2:43, 1 user, load average: 0.00, 0.00, 0.00
```

### >finger:

Command is a user information look-up command which gives details of all the users who are logged in. This tool is generally used by system administrators, which give a information like: login name, user name and terminal name, idle time, login time and in some cases email address also.

```
$ finger
```

Login	Name	Tty	Idle	Login Time	Office	Office Phone
exam1		pts/0		Apr 1 02:26	(192.168.1.248)	

### >cal:

Command is a calendar command which is used to see the calendar of a specific month or whole year

```
$ cal
```

```
[exam1@localhost ~]$ cal
```

```
April 2022
```

```
Su Mo Tu We Th Fr Sa
```

```
1 2 3 4 5 6
```

```
7 8 9 10 11 12 13
```

```
14 15 16 17 18 19 20
```

```
21 22 23 24 25 26 27
```

```
28 29 30
```

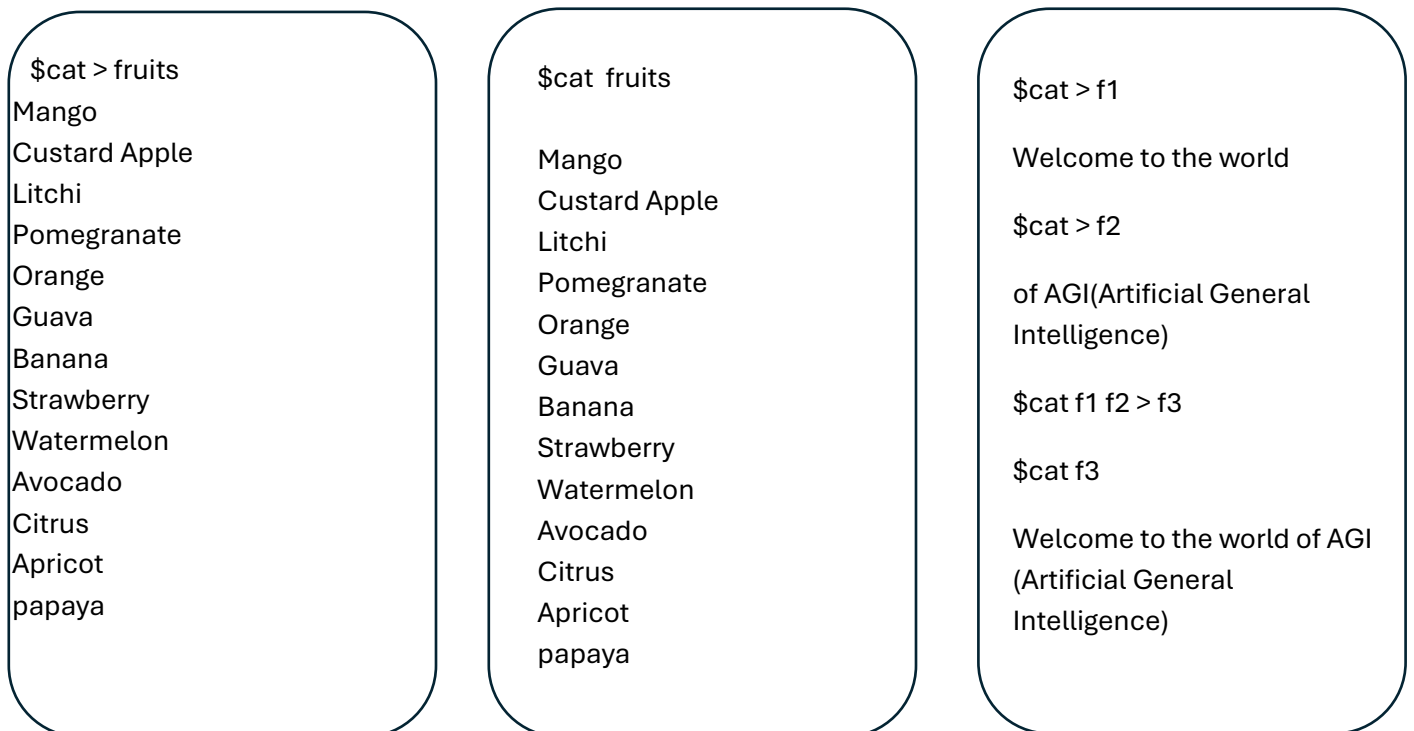
```
$ cal -y
```

```
[whole year calendar]
```

**(b) To display the contents of file using : cat, vi, more, head, tail, grep, cmp, wc :**

**>cat:** command is most frequently used UNIX command which lists, combines and writes (create) file content to the standard output. There are different ways to use 'cat' command:

- \$ cat > filename: creates a new file
- \$cat filename: displays content of a file
- \$cat filename1 filename2 > filename3 : merges filename1 and filename2 and stores the output in filename3.



**>vi:** 'vi' uses 2 operating modes to work:

- Insert mode
- Command mode
- Insert mode is used to create and edit files.
- Command mode: used to perform operations such as : saving, opening, copying a file etc.

```
$vi f3.txt
~
vegetables
green mango
tomato
brinjal
"f3.txt" [New File]
```

To insert, press [i]

To save and quit, press [ :wq]

**>more:** used to give the text file in the command prompt. It also allows the user to scroll up and down through the page.

```
$more f3.txt
vegetables
green mango
tomato
brinjal
```

**>head:** command allows us to view the first 10 lines of a text in the specified file top to bottom.

```
$head fruits

Mango
Custard Apple
Litchi
Pomegranate
Orange
Guava
Banana
Strawberry
Watermelon
Avocado
```

```
$head -5 fruits

Mango
Custard Apple
Litchi
Pomegranate
Orange
```

>**tail**: command allows us to view the last 10 lines of a text in the specified file bottom to top.

```
$tail fruits
```

```
Pomegranate  
Orange  
Guava  
Banana  
Strawberry  
Watermelon  
Avocado  
Citrus  
Apricot  
papaya
```

```
$tail -5 fruits
```

```
Watermelon  
Avocado  
Citrus  
Apricot  
papaya
```

>**grep**: command or global regular expression print command allows us to find a word by searching through all text in a specific file.

- Once a grep command finds a match, it prints all the lines that contains the specific pattern.
- This command helps to filter through a large log-files.

```
grep AGI f3
```

```
Welcome to the world of AGI(Artificial general intelligence)
```

>**cmp**: Command is used to compare the two files byte-by-byte to find out whether two files are identical or not.

- When 'CMP' command is used for comparison between two files, it reports the location of the first match to the screen, if difference is found. If not the files compared are identical.
- CMP displays no message, just simply returns the prompt when files are identical.

```
$ cat > file1  
Welcome to KLE
```

```
$cat > file2  
Welcome to KLE
```

```
$cat > file3  
All the best
```

```
$ cmp file1 file2  
cmp: EOF on file2 (no difference)
```

```
$cmp file2 file3  
file 2   file3   differ: byte1, line1
```

**>wc:** Stands for word count.

As name suggest, this command is used for counting purpose.

It is used to find number of lines, word count, byte / character count in the specified file.

By default it displays 4 columns.

The 1<sup>st</sup> column shows : number of lines.

The 2<sup>nd</sup> column shows: number of words.

The 3<sup>rd</sup> column shows: number of character.

The 4<sup>th</sup> column shows: filename.

```
$wc demo
```

```
5 10 63 demo
```

```
cat > demo
```

```
India Asia
```

```
USA America
```

```
China Asia
```

```
Sudan Africa
```

**c) To manage the files using: cat, cp, ls, rm, chmod, find, mv:**

**>cat:** To create, display the file.

```
$cat > numbers
One
Two
Three
```

**>cp:** Command is used to copy the files or directories and their contents.

- It will copy the content of a source file to a new destination file.

Syntax:

```
$ cp source_file designation_file
```

- It will copy by erasing existing destination.

```
$cat > f4

Destination file

$cp f4 f5

$cat f5

Destination file
```

**>ls:** Command lists all the files directories within a system.

-ls command with some option:

**\$ ls -a** : show all hidden files including visible ones.

**\$ ls -l** : displays long listing of all files

**\$ls -R**: displays all files, directories and sub-directories.

**\$ls -lh** : shows file sizes in easily readable formats such as: kb, mb, gb

**\$ls -o** : shows other files

```
$ls

demo f1 f2 f3 f4 country fruits
vegetables
```



**>rm:** command is used to delete the files within a directory.

To remove multiple files the:

Syntax: `$ rm file1 file2 ...`

```
$rm demo
```

```
$ls
```

```
f1 f2 f3 f4 country fruits vegetables
```

**>chmod:** Change the mode of permission of file. Chmod is a command that modifies a file / directories read, write and execute permissions.

- In unix, each file is associated with 3 user-classes.
- Owner
- Group members ( group of users)
- Others

Syntax: `$ chmod permission filename`

```
$chmod 777 demo
```

```
$ls -l demo
```

```
-rwxrwxrwx. 1 mca mca 0 Nov 15 15:24 demo
```

**>find:** used to find particular files and directories. It supports searching by filename, directory name, creation date, modification date and permissions etc.

```
$find f1
```

```
f1
```

**>mv:** used to move and rename the files and directories.

Syntax: \$ mv    source\_file    destination\_file  
          (old file)            (new file)

```
$cat > f6
```

Demo of mv command

```
$mv f6 f6.txt
```

```
$cat f6
```

cat: f6: No such file or directory

```
$cat f7
```

Demo of mv command

**>mkdir:** command used to create directories

```
$mkdir demodr
```

```
$ls
```

```
demodr demo f1 f2 f3 f4
```

**(d) Process utilities using: ps, pid, ppid, tty, time, kill, exit.**

**>ps:** ps stands for process status.

- Ps command produces a snapshot of all running processes in the system.
- The unique process id(PID) , the type of terminal (TTY) , the running time (TIME) , the command that launches the process (CMD)

**\$ ps**

PID	tty	TIME	CMD
3937	pts/9	00:00:00	tcsh
4248	pts/9	00:00:00	ps

**>PID:** is an acronym for Process Identification number on a linux / unix like OS.

- A pid is automatically assigned to each process when it is created.
- Process is nothing but running instance of a program and each process has a unique pid on a unix like system.

**>PPID :** It is the PID of the process present.

Eg: Process-2 with a PIN of 101 starts a process name-2 then process-2 will be given unique PID, such as 3240.

**\$ ps -e**

UID	PID	PPID	C	TIME	TTY	TIME
Soundart	8390	3372	0	19:14	pts/0	00:00:00
Soundart	3461	3390	0	19:25	pts/0	00:00:00

**>TTY:** It is a short for teletype, but popularly known as terminal, device is character-by-character and the communication between to them is consted by the TTY interface.

**\$ tty**

/dev/pts/9

>**TIME**: used to execute a command and prints a summary of real-time, user CPU time and system CPU time spent by executing a command when it terminates.

- 'real' time is the time elapsed wall clock time taken by a command to get executed.

- while 'user' time 'sys' time are the number of CPU time that command uses in user and kernel mode respectively.

```
$ time
```

```
Real    0m0.000s
```

```
User    0m0.000s
```

```
Sys     0m0.000s
```

>**kill**: use the kill command to terminate an unresponsive program manually.

It will signal misbehaving applications and instruct them to close their processes.

```
$ ps ux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY
User1	3645	0.0	0.1	5320	1780	pts/1
User2	3847	0.0	0.1	5332		

```
$kill 3645
```

```
$ps ux
```

User2	3847	0.0	0.1	5332		
-------	------	-----	-----	------	--	--

### (e) Directory handling utilities using: mkdir, cd, rmdir, pwd, mv

>**mkdir** : To create a new directory

>**cd**: to change other directory to current directory to use

>**rmdir**: to remove existing directory

```
$mkdir dir1  
  
$cd dir1  
  
$cd..[to come out of directory  
  
$rmdir dir1  
  
$cd dir1  
  
cd: dir1: No such file or directory
```

>**pwd**: stands for print working directory.

It writes full pathname of your current directory ( from the root)

```
$pwd  
  
/home/mca
```

>**mv**: used for moving a file or directory from source to destination.

```
$mkdir dir2  
$mv dir2 demodir  
$cd dir2  
  
cd: dir2: No such file or directory
```

**2. Write a shell script that displays list of all the files in the current directory to which the user has read, write and execute permissions.**

**\$nano program2. sh**

```
echo "the list of files with read, write and execution permissions"
for file in *
do
if [ -f $file ]
then
if [ -r $file -a -w $file -a -x $file ]
then
ls -l $file
fi
fi
done
```

Output:

```
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$ sh program2.sh
the list of files with read, write and execution permissions
-rwxrwxr-x 1 mca mca 36 Oct 23 10:20 demo
```

**3. Write a shell script that accepts a list of file names as its arguments, count and reports the occurrence of each word that is present in the first argument file on other argument files.**

**\$nano prg3.sh**

```
echo "enter the number of files:"
```

```
read n
```

```
echo "enter the "n" files:"
```

```
read fn
```

```
set $fn
```

```
for i in `cat $1`
```

```
do
```

```
echo -e " word = $i"
```

```
echo -e "-----"
```

```
grep -c "$i" $*
```

```
echo -e "-----"
```

```
done
```

**Output:**

**mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/part4\$ sh program3.sh**

**Enter the number of files:**

**3**

**Enter the 3 files :**

**demo test1.txt test2.txt**

**word = vegetables**

**-----**

**demo:1**

**test1.txt:0**

**test2.txt:0**

**-----**

**word = tomato**

**-----**

**demo:1**

**test1.txt:0**

**test2.txt:0**

-----

**word = turnip**

-----

**demo:1**

**test1.txt:0**

**test2.txt:0**

-----

**word = cabbage**

-----

**demo:1**

**test1.txt:0**

**test2.txt:0**

-----

**demo:1**

**test1.txt:0**

**test2.txt:0**

-----



**4. 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.**

**\$nano program4.sh**

```
echo " enter the file name :"  
read file  
set $file  
for x in $file  
do  
if [ ! -f$x ]  
then  
echo " file not found "  
continue  
fi  
tr '[a-z]' '[A-Z]' < $x  
done
```

**Output:**

**\$cat > abc**

abcdefgh

**\$sh program4.sh**

Enter the file name: abc

ABCDEFGH

**5. Write grep commands to the following:**

**a. To select the lines from a file that has exactly 2 characters.**

**b. To select the lines from a file that has more than one blank spaces.**

**a.**

```
$nano program5a.sh
```

```
fi
```

```
if
```

```
done
```

```
print
```

```
grep -c -E "^.{2}$" program5a.sh
```

**Output:**

```
fi
```

```
if
```

**b.**

```
$nano program5b.sh
```

```
Echo "Welcome tothenewworld"
```

**Output:**

```
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$ nano program8.sh
```

```
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$ nano program5b.sh
```

```
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$ grep " . *" program5b.sh
```

```
echo "Welcome to new world"
```

**6. Write a shell script which accepts two file names as arguments. Compare the contents. If they are same, then delete the second file.**

**\$nano program6.sh**

```
echo "enter the first file name"
read file1
echo "enter the second file name"
read file2
if [ -f $file1 ]
then
    if cmp $file1 $file2
    then
        echo " the content are the same deleting the second file"
        rm $file2
    else
        echo "the content are different"
    fi
else
    echo "file not found"
fi
```

**Output:**

```

[exam1@localhost ~]$ cat xy.txt
Welcome to UNIX programming
[exam1@localhost ~]$ cat yz.txt
Welcome to UNIX programming
[exam1@localhost ~]$ cat program6.sh
echo "enter the first file name"
read file1
echo "enter the second file name"
read file2
if [ -f $file1 ]
then
    if cmp $file1 $file2
    then
        echo " the content are the same deleting the second file"
        rm $file2
    else
        echo "the content are different"
    fi
else
    echo "file not found"
fi
[exam1@localhost ~]$ sh program6.sh
enter the first file name
xy.txt
enter the second file name
yz.txt
the content are the same deleting the second file
[exam1@localhost ~]$ _

```

## 7. Write a shell script

- a. to count number of lines in a file that do not contain vowels.
- b. to count number of characters, words, lines in a given file.

- a. To count number of lines that do not contain vowels.

**\$nano program7a.sh**

```

echo "enter the filename"
read file
count=0
awk '$0!~/[aeiou]/{ count++ }
END{printf "the number of lines without vowels are : %d\n", count}' $file

```

## Output:

**\$ sh program7.sh**

**enter the filename**

**xy.txt**

**the number of lines without vowels are : 1**

**\$ cat xy.txt**

Welcome to UNIX programming

THROUGH

**b. to count number of characters, words, lines in a given file.**

```
$nano program7b.sh
echo "enter the filename:"
read file
chars=$(wc -m < "$file")
words=$(wc -w < "$file")
lines=$(wc -l < "$file")
echo "the file has $chars characters, $words words and $lines lines"
```

**Output:**

```
$ sh program7b.sh
enter the filename:
xy.txt
the file has 43 characters, 5 words and 2 lines
[exam1@localhost ~]$ cat xy.txt
Welcome to UNIX programming
THRVIKLGHAEIOU
```

**8. Write a shell script to list all the files in the directory**

**nano program8.sh**

```
echo "enter the directory name"
read dir
for file in "${dir}"/*
do
echo $file
done
```

**Output:**

```
[exam1@localhost ~]$ sh program8.sh
```

```
enter the directory name
program
programone.txt
programthree.txt
programtwo.txt
```

9. Write a shell script to read three text files in the current directory and merge them into a single file and returns a file descriptor for the new file.

**\$nano program9.sh**

```
if [ $# -eq 3 ]
```

```
then
```

```
cat $1 $2 $3 > merged.txt
```

```
echo "files merged and named merged.txt"
```

```
else
```

```
echo "supply exactly three files"
```

```
fi
```

```
[exam1@localhost ~]$ sh program9.sh file01.txt file02.txt file03.txt
```

```
files merged and named merged.txt
```

```
[exam1@localhost ~]$ cat merged.txt
```

```
Welcome
```

```
to the world
```

```
of programming
```

```
[exam1@localhost ~]$ cat program9.sh
```

```
if [ $# -eq 3 ]
```

```
then
```

```
cat $1 $2 $3 > merged.txt
```

```
echo "files merged and named merged.txt"
```

```
else
```

```
echo "supply exactly three files"
```

```
fi
```

```
[exam1@localhost ~]$ cat file01.txt
```

```
Welcome
```

```
[exam1@localhost ~]$ cat file02.txt
```

```
to the world
```

```
[exam1@localhost ~]$ cat file03.txt
```

```
of programming
```

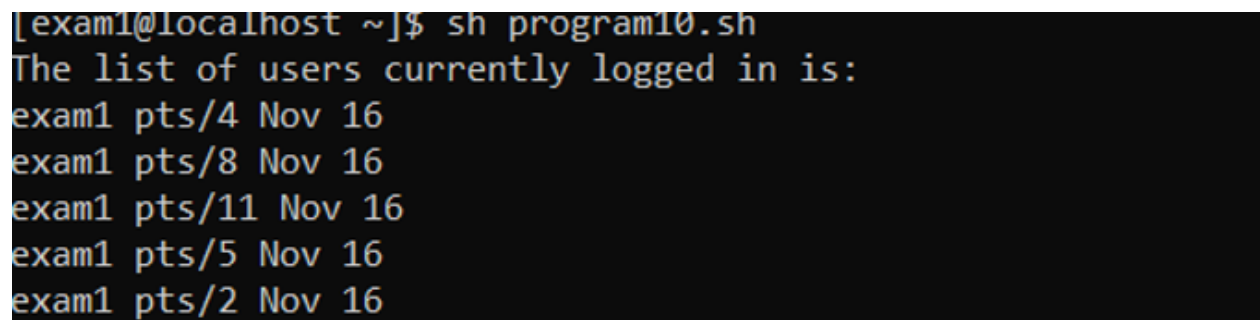
```
[exam1@localhost ~]$
```

**10. Write a shell script to display list of users currently logged in**

**\$nano program10.sh**

```
#!/bin/bash
# This script prints the list of users currently logged in
echo "The list of users currently logged in is:"
who | awk '{print $1, $2, $3, $4}'
```

**Output:**

A terminal window with a black background and yellow text. The prompt is [exam1@localhost ~]. The user enters 'sh program10.sh'. The output is 'The list of users currently logged in is:' followed by five lines of user login information: 'exam1 pts/4 Nov 16', 'exam1 pts/8 Nov 16', 'exam1 pts/11 Nov 16', 'exam1 pts/5 Nov 16', and 'exam1 pts/2 Nov 16'.

```
[exam1@localhost ~]$ sh program10.sh
The list of users currently logged in is:
exam1 pts/4 Nov 16
exam1 pts/8 Nov 16
exam1 pts/11 Nov 16
exam1 pts/5 Nov 16
exam1 pts/2 Nov 16
```

### 11. Write a program to copy a file into another using system calls.

**\$nano program11.c**

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

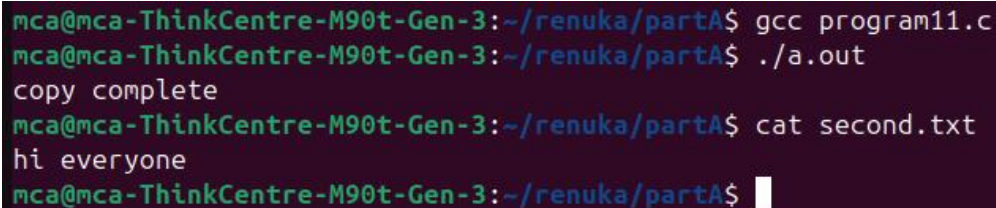
void main()
{
    char buf;
    int file_one, file_two;

    file_one = open("one.txt", O_RDONLY);
    if(file_one == -1)
    {
        printf("Error opening file \n");
        close(file_one);
        return;
    }
    file_two = open("second_file.txt", O_WRONLY);
    while(read(file_one, &buf, 1))
    {
        write(file_two, &buf, 1);
    }
    printf("Copy complete\n");
    close(file_one);
    close(file_two);
}
```

#### Output:

**\$cat > second.txt**

**This is a demo to demonstrate cp command**



```
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$ gcc program11.c
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$ ./a.out
copy complete
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$ cat second.txt
hi everyone
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$
```



**12. Write a program using system call: create, open, write, close, stat, fstat, lseek.**  
**\$nano program12.c**

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>

void main()
{
    char buf[5];
    int c = creat("sample_file.txt", O_WRONLY);
    int a = open("sample_file.txt", O_WRONLY);
    write(a, "Unix Lab", strlen("UNIX Lab"));
    close(a);
    int b = open("sample_file.txt", O_RDONLY);
    lseek(b, 5, SEEK_CUR);
    read(b, buf, 3);
    printf("characters read: %s\n", buf);
    struct stat s;
    stat("sample_file.txt", &s);
    printf("st_mode=%o\n", s.st_mode);
    printf("st_mtime=%ld\n", s.st_mtime);
}
```

```
mca@mca-ThinkCentre-M90t-Gen-3:~/reuka/partA$ gcc program12.c
mca@mca-ThinkCentre-M90t-Gen-3:~/reuka/partA$ ./a.out
characters read:
st_mode=100001
st_mtime=1729827993
mca@mca-ThinkCentre-M90t-Gen-3:~/reuka/partA$
```

**13. Write a program to create a child process and allow the parent to display "parent" and the child to display "child" on the screen.**

**\$nano program13.c**

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    int pid =fork();
    if (pid > 0)
    {
        printf("Parent process with id: %d\n", getpid());
    }
    else if(pid==0)
    {
        printf("child process with id: %d\n", getpid());
    }
    else
    {
        printf("fork() did not work");
    }
    return 0;
}
```

**Output:**

```
mca@mca-ThinkCentre-M90t-Gen-3:~/reuka/partA$ gcc program13.c
mca@mca-ThinkCentre-M90t-Gen-3:~/reuka/partA$ ./a.out
parent process with id: 3344
child process with id: 3345
```

**14. Write a program to create a Zombie process.**

**\$nano program14.c**

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main()
{
int pid = fork();
if(pid > 0)
{
printf("parent process going to sleep for next 25 seconds \n");
sleep(25);
}
else
{
printf("child process exiting \n");
exit(0);
}
return 0;
}
```

**Output:**

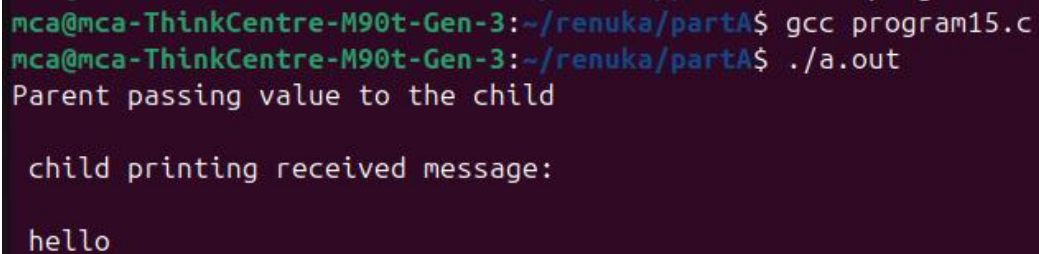
```
mca@mca-ThinkCentre-M90t-Gen-3:~/reuka/partA$ gcc program14.c
mca@mca-ThinkCentre-M90t-Gen-3:~/reuka/partA$ ./a.out
3366child process exiting
0parent process going to sleep for next 25 second
```

**15. Write a program to implement inter process communication using pipes**  
**\$nano program15.c**

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>

int main()
{
    int fd[2],n;
    char buf[10];
    pid_t P;
    pipe(fd);
    P = fork();
    if (P > 0)
    {
        printf("Parent passing value to the child \n");
        write(fd[1]," \n hello \n", 10);
        wait(NULL);
    }
    else
    {
        printf("\n child printing received message:\n");
        n=read(fd[0],buf,10);
        write(1,buf,n);
    }
}
```

**Output:**



```
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$ gcc program15.c
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$ ./a.out
Parent passing value to the child

child printing received message:
hello
```

## 16. Simulate the following CPU scheduling algorithms

### a. RoundRobin

### b.SJF

### Roundrobin

#### **\$nano program16.c**

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>

int main()
{
    int i, limit, total = 0, x, counter = 0, time_quantum;
    int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10];
    float average_wait_time, average_turnaround_time;
    printf("\n Enter Total Number of Processes:\t");
    scanf("%d", &limit);
    x = limit;
    for(i = 0; i < limit; i++)
    {
        printf("\n Enter Details of Process[%d]\n", i + 1);

        printf("Arrival Time:\t");

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

        printf("Burst Time:\t");

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

        temp[i] = burst_time[i];
    }

    printf("\nEnter Time Quantum:\t");
    scanf("%d", &time_quantum);
    printf("\nProcess ID \t \t Burst Time \t Turnaround Time \t Waiting Time \n");
    for(total = 0, i = 0; x != 0;)
    {
        if(temp[i] <= time_quantum && temp[i] > 0)
        {
            total = total + temp[i];
            temp[i] = 0;
            counter = 1;
        }
        else if(temp[i] > 0)
        {
            temp[i] = temp[i] - time_quantum;
            total = total + time_quantum;
        }
    }
}
```

```

    }
    if(temp[i] == 0 && counter == 1)
    {
        x--;
        printf("\nProcess[%d]\t\t%d \t\t %d \t\t %d\n", i + 1, burst_time[i], total - arrival_time[i],
total - arrival_time[i] - burst_time[i]);
        wait_time = wait_time + total - arrival_time[i] - burst_time[i];
        turnaround_time = turnaround_time + total - arrival_time[i];
        counter = 0;
    }
    if(i == limit - 1)
    {
        i = 0;
    }
    else if(arrival_time[i + 1] <= total)
    {
        i++;
    }
    else
    {
        i = 0;
    }
}

average_wait_time = (wait_time * 1.0 / limit);
average_turnaround_time = (turnaround_time * 1.0 / limit);
printf("\n Average Waiting Time: %f", average_wait_time);
printf("\n Avg Turnaround Time: %f", average_turnaround_time);
return 0;
}

```

```
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$ gcc program16.c
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$ ./a.out
```

Enter Total Number of Processes: 5

Enter Details of Process[1]

Arrival Time: 0

Burst Time: 20

Enter Details of Process[2]

Arrival Time: 1

Burst Time: 8

Enter Details of Process[3]

Arrival Time: 2

Burst Time: 15

Enter Details of Process[4]

Arrival Time: 3

Burst Time: 9

Enter Details of Process[5]

Arrival Time: 4

Burst Time: 2

Enter Time Quantum: 4

Process ID	Burst Time	Turnaround Time	Waiting Time
Process[5]	2	14	12
Process[2]	8	25	17
Process[4]	9	40	31
Process[3]	15	48	33
Process[1]	20	54	34

Average Waiting Time: 25.400000

Avg Turnaround Time: 36.200001mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA\$

## **SJF**

### **\$ program16\_SJF.c**

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>

void main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter number of process:");
    scanf("%d",&n);

    printf("\nEnter Burst Time:\n");
    for(i=0;i<n;i++)
    {
        printf("p%d:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1;    //contains process number
    }

    //sorting burst time in ascending order using selection sort
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }

        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;

        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }

    wt[0]=0;    //waiting time for first process will be zero

    //calculate waiting time
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
```



```

        wt[i]+=bt[j];

    total+=wt[i];
}

avg_wt=(float)total/n;    //average waiting time
total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];    //calculate turnaround time
    total+=tat[i];
    printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
}

avg_tat=(float)total/n;    //average turnaround time
printf("\n\nAverage Waiting Time=%f",avg_wt);
printf("\nAverage Turnaround Time=%f\n",avg_tat);
}

```

### Output:

```

mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$ gcc program16_SJF.c
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$ ./a.out
Enter number of process:5

Enter Burst Time:
p1:3
p2:5
p3:1
p4:6
p5:8

Process      Burst Time      Waiting Time      Turnaround Time
p3           1              0                1
p1           3              1                4
p2           5              4                9
p4           6              9               15
p5           8             15               23

Average Waiting Time=5.800000
Average Turnaround Time=10.400000
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$

```

## 17. Write a program that illustrates file locking using semaphores.

**\$nano program17.c**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>

int main()
{
    int fd;
    struct flock lock;
    fd = open("three.txt",O_WRONLY);
    if(fd == -1)
    {
        printf("Error opening file");
        return;
    }
    memset(&lock,0,sizeof(lock));
    lock.l_type=F_WRLCK;
    lock.l_whence = SEEK_SET;
    lock.l_start = 0;
    lock.l_len=0;

    if(fcntl(fd,F_SETLK, &lock) == -1)
    {
        perror("Failed to acquire lock");
        exit(1);
    }
    dprintf(fd,"Locked file \n");

    lock.l_type=F_UNLCK;
    if(fcntl(fd,F_SETLK, &lock) == -1)
    {
        perror("Failed to release lock");
        exit(1);
    }
    close(fd);
    return 0;
}
```

## Output:

```
mca@mca-ThinkCentre-M90t-Gen-3:~/reuka/partA$ gcc program17.c
program17.c: In function 'main':
program17.c:15:1: warning: 'return' with no value, in function returning non-void
   15 | return;
      | ^~~~~~
program17.c:7:5: note: declared here
     7 | int main()
      |     ^~~~~
mca@mca-ThinkCentre-M90t-Gen-3:~/reuka/partA$ ./a.out
mca@mca-ThinkCentre-M90t-Gen-3:~/reuka/partA$ cat three.txt
Locked file
```

**18. Write a program that implements a producer-consumer system with two processes (using semaphores)**

**\$nano program18.c**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <sys/types.h>
#include <sys/ipc.h>
// C program for the above approach
```

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

```
// Initialize a mutex to 1
int mutex = 1;
```

```
// Number of full slots as 0
int full = 0;
```

```
// Number of empty slots as size
// of buffer
int empty = 10, x = 0;
```

```
// Function to produce an item and
// add it to the buffer
```

```
void producer()
{
    // Decrease mutex value by 1
    --mutex;
    ++full;
    --empty;
    x++;
    printf("\nProducer produces"
           "item %d",
           x);

    // Increase mutex value by 1
    ++mutex;
}
```

```
// Function to consume an item and
// remove it from buffer
```

```
void consumer()
{
    // Decrease mutex value by 1
```

```

--mutex;
--full;
++empty;
printf("\nConsumer consumes "
      "item %d",
      x);
x--;

// Increase mutex value by 1
++mutex;
}

// Driver Code
int main()
{
    int n, i;
    printf("\n1. Press 1 for Producer"
          "\n2. Press 2 for Consumer"
          "\n3. Press 3 for Exit");
#pragma omp critical

    for (i = 1; i > 0; i++) {

        printf("\nEnter your choice:");
        scanf("%d", &n);

        // Switch Cases
        switch (n) {
        case 1:
            if ((mutex == 1)
                && (empty != 0)) {
                producer();
            }

            else {
                printf("Buffer is full!");
            }
            break;

        case 2:
            if ((mutex == 1)
                && (full != 0)) {
                consumer();
            }
            else {
                printf("Buffer is empty!");
            }
            break;

```

```
        // Exit Condition
        case 3:
            exit(0);
            break;
    }
}
```

### Output:

```
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$ gcc program18.c
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$ ./a.out

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 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

Producer produces item 4
Enter your choice:2

Consumer consumes item 4
Enter your choice:2

Consumer consumes item 3
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!
Enter your choice:3
mca@mca-ThinkCentre-M90t-Gen-3:~/renuka/partA$
```

**19. Write a program that illustrates inter process communication using shared memory system calls**

**\$nano shared\_memory\_program**

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>

#define BUF_SIZE 1024
#define SHM_KEY 0x1234

struct shmseg {
    int cnt;
    int complete;
    char buf[BUF_SIZE];
};

int fill_buffer(char *bufptr, int size);

void writer() {
    int shmid, numtimes;
    struct shmseg *shmp;
    char *bufptr;
    int spaceavailable;

    // Create the shared memory segment
    shmid = shmget(SHM_KEY, sizeof(struct shmseg), 0644 | IPC_CREAT);
    if (shmid == -1) {
        perror("Writer: Shared memory creation failed");
        exit(1);
    }

    // Attach to the segment to get a pointer to it
    shmp = shmat(shmid, NULL, 0);
    if (shmp == (void *)-1) {
        perror("Writer: Shared memory attachment failed");
        exit(1);
    }

    // Initialize the shared memory segment
    shmp->cnt = 0;
    shmp->complete = 0;
    printf("Writer: Attached to shared memory, starting to write\n");
```

```

// Write data to the shared memory segment
bufptr = shmp->buf;
spaceavailable = BUF_SIZE;
for (numtimes = 0; numtimes < 5; numtimes++) {
    shmp->cnt = fill_buffer(bufptr, spaceavailable);
    printf("Writer: Writing iteration %d\n", numtimes + 1);
    printf("Writer: Wrote %d bytes, cnt: %d, complete: %d\n", shmp->cnt, shmp->cnt, shmp-
>complete);
    bufptr = shmp->buf;
    spaceavailable = BUF_SIZE;
    sleep(1);
}
shmp->complete = 1; // Indicate that writing is done
printf("Writer: Completed Writing. Final cnt: %d, complete: %d\n", shmp->cnt, shmp-
>complete);

// Detach from the shared memory segment
if (shmdt(shmp) == -1) {
    perror("Writer: Shared memory detachment failed");
    exit(1);
}
printf("Writer: Detached from shared memory\n");

// Remove the shared memory segment
if (shmctl(shmid, IPC_RMID, 0) == -1) {
    perror("Writer: Shared memory control operation failed");
    exit(1);
}
printf("Writer: Complete\n");
}

void reader() {
    int shmid;
    struct shmseg *shmp;

    // Create the shared memory segment
    shmid = shmget(SHM_KEY, sizeof(struct shmseg), 0644 | IPC_CREAT);
    if (shmid == -1) {
        perror("Reader: Shared memory creation failed");
        exit(1);
    }

    // Attach to the segment to get a pointer to it
    shmp = shmat(shmid, NULL, 0);
    if (shmp == (void *)-1) {
        perror("Reader: Shared memory attachment failed");
        exit(1);
    }
}

```



```

}
printf("Reader: Attached to shared memory\n");

// Read data from the shared memory segment
while (1) {
    if (shmp->cnt > 0) {
        printf("Reader: Segment contains: \"%s\"\n", shmp->buf);
        printf("Reader: Read %d bytes\n", shmp->cnt);
    }
    printf("Reader: Debug - cnt: %d, complete: %d\n", shmp->cnt, shmp->complete);
    sleep(1);

    if (shmp->complete == 1) { // Check if the writer has completed writing
        printf("Reader: Detected complete flag\n");
        break;
    }
}
printf("Reader: Reading Done, Detaching Shared Memory\n");

// Detach from the shared memory segment
if (shmdt(shmp) == -1) {
    perror("Reader: Shared memory detachment failed");
    exit(1);
}
printf("Reader: Complete\n");
}

int fill_buffer(char *bufptr, int size) {
    static char ch = 'A';
    int filled_count;

    memset(bufptr, ch, size - 1);
    bufptr[size - 1] = '\0';
    if (ch > 122) {
        ch = 65;
    }
    if ((ch >= 65) && (ch <= 122)) {
        if ((ch >= 91) && (ch <= 96)) {
            ch = 65;
        }
    }
    filled_count = strlen(bufptr);
    ch++;
    return filled_count;
}

int main(int argc, char *argv[]) {
    if (argc != 2) {

```

```

    fprintf(stderr, "Usage: %s -w | -r\n", argv[0]);
    exit(1);
}

if (strcmp(argv[1], "-w") == 0) {
    writer();
} else if (strcmp(argv[1], "-r") == 0) {
    reader();
} else {
    fprintf(stderr, "Invalid option. Use -w for writer and -r for reader.\n");
    exit(1);
}

return 0;
}

```

### Output:

#run both `./shared_memory_program -w` & `./shared_memory_program -r` in two terminals simultaneously

`gcc -o shared_memory_program shared_memory_program.c`  
`./shared_memory_program -w`

```

mca@mca-ThinkCentre-M90t-Gen-3:~$ ./shared_memory_program -w
Writer: Attached to shared memory, starting to write
Writer: Writing iteration 1
Writer: Wrote 1023 bytes, cnt: 1023, complete: 0
Writer: Writing iteration 2
Writer: Wrote 1023 bytes, cnt: 1023, complete: 0
Writer: Writing iteration 3
Writer: Wrote 1023 bytes, cnt: 1023, complete: 0
Writer: Writing iteration 4
Writer: Wrote 1023 bytes, cnt: 1023, complete: 0
Writer: Writing iteration 5
Writer: Wrote 1023 bytes, cnt: 1023, complete: 0
Writer: Completed Writing. Final cnt: 1023, complete: 1
Writer: Detached from shared memory
Writer: Complete
mca@mca-ThinkCentre-M90t-Gen-3:~$ █

```

```
mca@nca-ThinkCentre-M90t-Gen-3: $ ./shared_memory_program -r
Reader: Attached to shared memory
Reader: Debug - cnt: 0, complete: 0
Reader: Debug - cnt: 0, complete: 0
Reader: Debug - cnt: 0, complete: 0
Reader: Debug - cnt: 0, complete: 0
Reader: Segment contains: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
Reader: Read 1023 bytes
Reader: Debug - cnt: 1023, complete: 0
Reader: Segment contains: "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
Reader: Read 1023 bytes
Reader: Debug - cnt: 1023, complete: 0
```

[illegible]

**20. Write a program that illustrates the following:**

- a. Creating message queue.**
- b. Writing to a message queue**
- c. Reading from a message queue**

**\$nano program20.c**

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MAX 10

// structure for message queue
struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;

    // ftok to generate unique key
    key = ftok("progfile", 65);

    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);
    message.mesg_type = 1;

    printf("Write Data : ");
    fgets(message.mesg_text, MAX, stdin);

    // msgsnd to send message
    msgsnd(msgid, &message, sizeof(message), 0);

    // display the message
    printf("Data sent is : %s \n", message.mesg_text);

    return 0;
}
```

**Output:**

**gcc program20.c**

**./a.out**

**Write Data : bhaani vani renu**

**Data sent is : bhaani vani renu**