OPERATING SYSTEM

Lab File

Submitted To:
Dr Abhishek Singhal

Submitted By:
Harsh Verma

OS Lab

A2305220154

CSE202

4CSE1-Y

Amity School of Engineering and Technology (ASET)
Amity University
Uttar Pradesh

List Of Experiments:

1. **Use of Basic UNIX Shell Commands/Linux Commands.**
2. **Commands related to inode, I/O redirection and piping, process control commands, mails**
3. **Shell Programming: Shell script exercises based on following:**
   - **(i) Interactive shell scripts**
   - **(ii) Positional parameters**
   - **(iii) Arithmetic**
   - **(iv) if-then-fi, if-then- else-fi, nested if-else**
   - **(v) Logical operators**
   - **(vi) else + if equals elif, case structure**
   - **(vii) while, until, for loops, use of break**
4. **Write a shell script that accept a file name starting and ending line numbers as arguments and display all the lines between given line no.**
5. **Write a shell script that delete all lines containing a specified word.**
6. **Write a shell script that displays a list of all the files in the current directory**
7. **Simulation of Unix commands using C.**
8. **Implement the following CPU Scheduling Algorithms.**
   i) FCFS ii) Shortest Job First.
9. **Implement the following CPU Scheduling Algorithms.**
   i) Round Robin ii) priority based
10. **Write a program to implement banker's algorithm.**
11. **Write a program to implement paging algorithm.**
12. **OPEN ENDED PROJECT**

Lab Practical-1

Aim: Use of Basic UNIX Shell Commands/Linux Commands.

Software Used: Command Prompt.

Commands:

1. pwd stands for Print Working Directory. It prints the path of the working directory, starting from the root.

```
sysadmin@localhost:~$ pwd
/home/sysadmin
sysadmin@localhost:~$
```

2. man command in Linux is used to display the user manual of any command that we can run on the terminal.

```
sysadmin@localhost:~$ man echo
```

```
ECHO(1)                          User Commands                          ECHO(1)

NAME
       echo - display a line of text

SYNOPSIS
       echo [SHORT-OPTION]... [STRING]...
       echo LONG-OPTION

DESCRIPTION
       Echo the STRING(s) to standard output.

       -n     do not output the trailing newline

       -e     enable interpretation of backslash escapes

       -E     disable interpretation of backslash escapes (default)

       --help display this help and exit

       --version
              output version information and exit

 Manual page echo(1) line 1 (press h for help or q to quit)
```

3. The pwd Linux command prints the current working directory path, starting from the root (/).

```
sysadmin@localhost:~$ pwd
/home/sysadmin
```

4. The cd ("change directory") command is used to change the current working directory in Linux and other Unix-like operating systems.

   a. cd ~ : this command is used to change directory to the home directory.

```
sysadmin@localhost:~/Desktop$ cd ~
sysadmin@localhost:~$
```

   b. To move inside a subdirectory : to move inside a subdirectory in linux we use, cd [directory]

```
sysadmin@localhost:~$ cd Desktop
sysadmin@localhost:~/Desktop$
```

   c. cd .. : this command is used to move to the parent directory of current directory, or the directory one level up from the current directory. ".." represents parent directory.

```
sysadmin@localhost:~/Desktop$ cd ..
sysadmin@localhost:~$
```

5. The ls command is used to list files or directories in Linux and other Unix-based operating systems.

```
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

   a. ls -l long list (displays lots of info)

```
sysadmin@localhost:~$ ls -l
total 32
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Desktop
drwx------ 4 sysadmin sysadmin 4096 Dec 20  2017 Documents
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Downloads
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Music
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Pictures
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Public
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Templates
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Videos
```

   b. ls -t sort by modification time

```
sysadmin@localhost:~$ ls -t
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

   c. ls -s sort by size

```
sysadmin@localhost:~$ ls -s
total 32
4 Desktop    4 Downloads  4 Pictures  4 Templates
4 Documents  4 Music      4 Public    4 Videos
```

   d. -h list file sizes in human readable format

```
sysadmin@localhost:~$ ls -h
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

   e. -r reverse the order

```
sysadmin@localhost:~$ ls -r
Videos  Templates  Public  Pictures  Music  Downloads  Documents  Desktop
```

NOTE: Options can be combined: “ls -ltr”

```
sysadmin@localhost:~$ ls -ltr
total 32
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Videos
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Templates
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Public
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Pictures
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Music
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Downloads
drwx------ 4 sysadmin sysadmin 4096 Dec 20  2017 Documents
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Desktop
```

6. asterisk, * , meaning "zero or more characters". When you type a command like ls a* , the shell finds all filenames in the current directory starting with a and passes them to the ls command.

```
sysadmin@localhost:~$ ls P*
Pictures:

Public:
```

7. mkdir command in Linux allows the user to create directories (also referred to as folders in some operating systems).

```
sysadmin@localhost:~/Documents$ mkdir Harsh
sysadmin@localhost:~/Documents$ ls
Harsh             alpha-second.txt  hello.sh      newhome.txt  red.txt
School            alpha-third.txt   hidden.txt    numbers.txt
Work              alpha.txt         letters.txt   os.csv
adjectives.txt    animals.txt       linux.txt     people.csv
alpha-first.txt   food.txt          longfile.txt  profile.txt
```

8. rmdir command is used remove empty directories from the filesystem in Linux. The rmdir command removes each and every directory specified in the command line only if these directories are empty.

```
sysadmin@localhost:~/Documents$ rmdir Harsh
sysadmin@localhost:~/Documents$ ls
School           alpha-second.txt  food.txt     linux.txt     os.csv
Work             alpha-third.txt   hello.sh     longfile.txt  people.csv
adjectives.txt   alpha.txt         hidden.txt   newhome.txt   profile.txt
alpha-first.txt  animals.txt       letters.txt  numbers.txt   red.txt
```

Lab Practical-2

Aim: Commands related to inode, I/O redirection and piping, process control commands, mails.

Software Used: Command Prompt.

Commands:

1. The VI editor is the most popular and classic text editor in the Linux family. Below, are some reasons which make it a widely used editor –
   1) It is available in almost all Linux Distributions
   2) It works the same across different platforms and Distributions
   3) It is user-friendly.

```
sysadmin@localhost:~/Documents$ vi
```

```
~
~
~                          VIM - Vi IMproved
~
~                            version 7.4.52
~                       by Bram Moolenaar et al.
~           Modified by pkg-vim-maintainers@lists.alioth.debian.org
~                 Vim is open source and freely distributable
~
~                        Help poor children in Uganda!
~                type  :help iccf<Enter>       for information
~
~                type  :q<Enter>               to exit
~                type  :help<Enter>  or  <F1>  for on-line help
~                type  :help version7<Enter>   for version info
~
~                        Running in Vi compatible mode
~                type  :set nocp<Enter>        for Vim defaults
~                type  :help cp-default<Enter> for info on this
~
~
~
```

```
sysadmin@localhost:~/Documents$ vi animals.txt
```

```
1 retriever
2 badger
3 bat
4 wolf
5 eagle
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"animals.txt" 5 lines, 42 characters
```

2. The cat command is a utility command in Linux. One of its most known usages is to print the content of a file onto the standard output stream.

```
sysadmin@localhost:~/Documents$ cat animals.txt
1 retriever
2 badger
3 bat
4 wolf
5 eagle
```

3. The head is a command present in all major Linux distributions which are used to print out data from the start of a file. It is the opposite of the tail command which is used to output data from the end of a file.

```
sysadmin@localhost:~/Documents$ head animals.txt
1 retriever
2 badger
3 bat
4 wolf
5 eagle
```

```
sysadmin@localhost:~/Documents$ tail animals.txt
1 retriever
2 badger
3 bat
4 wolf
5 eagle
```

4. 'cp' means copy. 'cp' command is used to copy a file or a directory.

```
sysadmin@localhost:~/Documents$ ls
School            alpha-second.txt  food.txt     linux.txt     os.csv
Work              alpha-third.txt   hello.sh     longfile.txt  people.csv
adjectives.txt    alpha.txt         hidden.txt   newhome.txt   profile.txt
alpha-first.txt   animals.txt       letters.txt  numbers.txt   red.txt
sysadmin@localhost:~/Documents$ cp animals.txt animals_new.txt
sysadmin@localhost:~/Documents$ ls
School            alpha-third.txt  hello.sh      newhome.txt  red.txt
Work              alpha.txt        hidden.txt    numbers.txt
adjectives.txt    animals.txt      letters.txt   os.csv
alpha-first.txt   animals_new.txt  linux.txt     people.csv
alpha-second.txt  food.txt         longfile.txt  profile.txt
```

5. mv stands for move. mv is used to move one or more files or directories from one place to another in a file system like UNIX.

```
sysadmin@localhost:~/Documents$ pwd
/home/sysadmin/Documents
sysadmin@localhost:~/Documents$ ls
School            alpha-third.txt  hello.sh      newhome.txt  red.txt
Work              alpha.txt        hidden.txt    numbers.txt
adjectives.txt    animals.txt      letters.txt   os.csv
alpha-first.txt   animals_new.txt  linux.txt     people.csv
alpha-second.txt  food.txt         longfile.txt  profile.txt
sysadmin@localhost:~/Documents$ mv animals_new.txt /home/sysadmin/Desktop
sysadmin@localhost:~/Documents$ cd ..
sysadmin@localhost:~$ cd Desktop
sysadmin@localhost:~/Desktop$ ls
animals_new.txt
```

6. The 'rm' means remove. This command is used to remove a file. The command line doesn't have a recycle bin or trash unlike other GUI's to recover the files.

```
sysadmin@localhost:~/Desktop$ ls
animals_new.txt
sysadmin@localhost:~/Desktop$ rm animals_new.txt
sysadmin@localhost:~/Desktop$ ls
sysadmin@localhost:~/Desktop$
```

7. Permission levels : "r" means "read only" permission
   "w" means "write" permission
   "x" means "execute" permission

   In case of directory, "**x**" grants permission to list directory contents

```
sysadmin@localhost:~$ ls -l
total 32
drwx------ 1 sysadmin sysadmin 4096 Apr  7 18:16 Desktop
drwx------ 1 sysadmin sysadmin 4096 Apr  7 18:10 Documents
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Downloads
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Music
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Pictures
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Public
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Templates
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Videos
```

8. To change directory permissions in Linux, use the following:
   chmod +rwx filename to add permissions.
   chmod -rwx directoryname to remove permissions.
   chmod +x filename to allow executable permissions.
   chmod -wx filename to take out write and executable permissions.
   Note that "r" is for read, "w" is for write, and "x" is for execute.

This only changes the permissions for the owner of the file.

```
sysadmin@localhost:~$ ls -l
total 32
drwx------ 1 sysadmin sysadmin 4096 Apr  7 18:16 Desktop
drwx------ 1 sysadmin sysadmin 4096 Apr  7 18:10 Documents
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Downloads
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Music
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Pictures
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Public
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Templates
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Videos
sysadmin@localhost:~$ chmod a+x Documents
sysadmin@localhost:~$ ls -l
total 32
drwx------ 1 sysadmin sysadmin 4096 Apr  7 18:16 Desktop
drwx--x--x 1 sysadmin sysadmin 4096 Apr  7 18:10 Documents
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Downloads
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Music
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Pictures
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Public
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Templates
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Videos
```

9. The cat command is a utility command in Linux. One of its most commonly known usages is to print the content of a file onto the standard output stream.

```
sysadmin@localhost:~/Documents$ cat hello.sh
printf ' _____________
( Hello World! )
 --------------
        \
         \
           <(^)
            ( )
'
```

10. ./ command is use to execute the content of a file.

```
sysadmin@localhost:~/Documents$ ./hello.sh
 _____________
( Hello World! )
 --------------
        \
         \
           <(^)
            ( )
```

11. ps command is used to list the currently running processes and their PIDs along with some other information depends on different options.

```
sysadmin@localhost:~/Documents$ ps
  PID TTY          TIME CMD
   87 pts/0    00:00:00 bash
  134 pts/0    00:00:00 ps
```

```
sysadmin@localhost:~/Documents$ ps -u root
  PID TTY          TIME CMD
    1 pts/0    00:00:00 init
   37 ?        00:00:00 cron
   39 ?        00:00:00 sshd
   77 pts/0    00:00:00 login
```

12. top command is used to show the Linux processes. It provides a dynamic real-time view of the running system.

```
top - 17:19:57 up 13 days,  2:23,  1 user,  load average: 0.01, 0.04, 0.00
Tasks:   8 total,   1 running,   7 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni, 99.9 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  98985536 total, 10680156 used, 88305376 free,  1485036 buffers
KiB Swap:  8388604 total,        0 used,  8388604 free.  3632844 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0   17980   2976   2724 S   0.0  0.0   0:00.03 init
   33 syslog    20   0  182128   2756   2332 S   0.0  0.0   0:00.39 rsyslogd
   37 root      20   0   23672   2152   1944 S   0.0  0.0   0:00.00 cron
   39 root      20   0   61400   3216   2540 S   0.0  0.0   0:00.00 sshd
   56 bind      20   0  893076  39580   5176 S   0.0  0.0   0:00.07 named
   77 root      20   0   63152   2988   2524 S   0.0  0.0   0:00.00 login
   87 sysadmin  20   0   18196   3340   2848 S   0.0  0.0   0:00.04 bash
  140 sysadmin  20   0   19872   2440   2120 R   0.0  0.0   0:00.00 top
```

13. The wc (word count) command in Unix/Linux operating systems is used to find out number of newline count, word count, byte and characters count in a files specified by the file arguments.

```
sysadmin@localhost:~/Documents$ wc animals.txt
 5 10 42 animals.txt
sysadmin@localhost:~/Documents$ cat animals.txt
1 retriever
2 badger
3 bat
4 wolf
5 eagle
sysadmin@localhost:~/Documents$
```

14. Piping

Command using Pipes	Meaning or Use of Pipes
$ ls | more	Output of ls command is given as input to more command So that output is printed one screen full page at a time.
$ who | sort	Output of who command is given as input to sort command So that it will print sorted list of users
$ who | sort > user_list	Same as above except output of sort is send to (redirected) user_list file
$ who | wc -l	Output of who command is given as input to wc command So that it will print number of user who logon to system
$ ls -l | wc -l	Output of ls command is given as input to wc command So that it will print number of files in current directory.

```
sysadmin@localhost:~/Documents$ ls | more
School
Work
a.sh
adjectives.txt
alpha-first.txt
alpha-second.txt
alpha-third.txt
alpha.txt
animals.txt
food.txt
harsh.sh
hello.sh
hidden.txt
letters.txt
linux.txt
longfile.txt
newhome.txt
nohup.out
numbers.txt
os.csv
people.csv
profile.txt
red.txt
```

```
sysadmin@localhost:~/Documents$ who | sort
sysadmin pts/0        Apr 10 17:22
```

```
sysadmin@localhost:~/Documents$ who | sort > user_list
sysadmin@localhost:~/Documents$ ls
School           alpha-second.txt  harsh.sh     longfile.txt  people.csv
Work             alpha-third.txt   hello.sh     newhome.txt   profile.txt
a.sh             alpha.txt         hidden.txt   nohup.out     red.txt
adjectives.txt   animals.txt       letters.txt  numbers.txt   user_list
alpha-first.txt  food.txt          linux.txt    os.csv
```

```
sysadmin@localhost:~/Documents$ ls
School           alpha-second.txt  food.txt     linux.txt     os.csv
Work             alpha-third.txt   hello.sh     longfile.txt  people.csv
adjectives.txt   alpha.txt         hidden.txt   newhome.txt   profile.txt
alpha-first.txt  animals.txt       letters.txt  numbers.txt   red.txt
sysadmin@localhost:~/Documents$ cat animals.txt
1 retriever
2 badger
3 bat
4 wolf
5 eagle
sysadmin@localhost:~/Documents$ cat animals.txt | wc
      5      10      42
sysadmin@localhost:~/Documents$ cat animals.txt | wc -w
10
sysadmin@localhost:~/Documents$ cat animals.txt | wc -c
42
sysadmin@localhost:~/Documents$ cat animals.txt | wc -l
5
```

15. To search files in a directory for a specific string use “grep”

```
sysadmin@localhost:~/Documents$ cat animals.txt
1 retriever
2 badger
3 bat
4 wolf
5 eagle
sysadmin@localhost:~/Documents$ grep "bat" animals.txt
3 bat
```

16. The alias command lets you give your own name to a command or sequence of commands. You can then type your short name, and the shell will execute the command or sequence of commands for you.
    $alias cls=clear

```
sysadmin@localhost:~/Documents$ alias clr=clear
sysadmin@localhost:~/Documents$ clr
```

17. The chown command allows you to change the owner and group owner of a file.
18. The curl command is a tool to retrieve information and files from Uniform Resource Locators (URLs) or internet addresses.
19. The finger command gives you a short dump of information about a user, including the time of the user’s last login, the user’s home directory, and the user account’s full name.
20. The history command lists the commands you have previously issued on the command line. You can repeat any of the commands from your history by typing an exclamation point ! and the number of the command from the history list.
21. The ping command lets you verify that you have network connectivity with another network device. It is commonly used to help troubleshoot networking issues. To use ping, provide the IP address or machine name of the other device.

22. You can obtain some system information regarding the Linux computer you're working on with the uname command.
    a. Use the -a (all) option to see everything.
    b. Use the -s (kernel name) option to see the type of kernel.
    c. Use the -r (kernel release) option to see the kernel release.
    d. Use the -v (kernel version) option to see the kernel version.

```
sysadmin@localhost:~/Documents$ uname -a
Linux localhost 4.15.0-173-generic #182-Ubuntu SMP Fri Mar 18 15:53:46 UTC 2022
x86_64 x86_64 x86_64 GNU/Linux
sysadmin@localhost:~/Documents$ uname -s
Linux
sysadmin@localhost:~/Documents$ uname -r
4.15.0-173-generic
sysadmin@localhost:~/Documents$ uname -v
#182-Ubuntu SMP Fri Mar 18 15:53:46 UTC 2022
```

Lab Practical-3

Aim: Shell Programming: Shell script exercises based on following:

(i) Interactive shell scripts

(ii) Positional parameters

(iii) Arithmetic

(iv) if-then-fi, if-then- else-fi, nested if-else

(v) Logical operators

(vi) else + if equals elif, case structure

(vii) while, until, for loops, use of break

Software Used: Command Prompt.

Commands:

```
sysadmin@localhost:~/Documents$ vi hello.sh
sysadmin@localhost:~/Documents$ cat hello.sh
echo "File name is "$0
echo $3
Data=$5
echo "A $Data costs is just $6."
echo $#
sysadmin@localhost:~/Documents$ bash hello.sh apple 5 banana 8 "Fruit Basket" 15
File name is hello.sh
banana
A Fruit Basket costs is just 15.
6
```

```
sysadmin@localhost:~/Documents$ vi hello.sh
sysadmin@localhost:~/Documents$ cat hello.sh
#!/bin/bash
echo "you executed this command: ${0}"
sysadmin@localhost:~/Documents$ bash hello.sh
you executed this command: hello.sh
```

```
sysadmin@localhost:~/Documents$ cat hello.sh
A=3
B=$((100 * $A + 5))
echo $B
sysadmin@localhost:~/Documents$ bash hello.sh
305
```

```
sysadmin@localhost:~/Documents$ vi hello.sh
sysadmin@localhost:~/Documents$ cat hello.sh
expr 1 + 3
expr 2 - 1
expr 10 / 2
expr 20 % 2
expr 10 \* 3
echo `expr 6 + 3`
sysadmin@localhost:~/Documents$ bash hello.sh
4
1
5
0
30
9
```

```
sysadmin@localhost:~/Documents$ vi hello.sh
sysadmin@localhost:~/Documents$ cat hello.sh
NAME="Bill"

if [ "$NAME" = "John" ]; then

  echo "True - my name is indeed John"

else

  echo "False"

  echo "You must mistaken me for $NAME"

fi
sysadmin@localhost:~/Documents$ bash hello.sh
False
You must mistaken me for Bill
```

```
sysadmin@localhost:~/Documents$ vi hello.sh
sysadmin@localhost:~/Documents$ cat hello.sh
NAME="George"

if [ "$NAME" = "John" ]; then

  echo "John Lennon"

elif [ "$NAME" = "George" ]; then

  echo "George Harrison"

else

  echo "This leaves us with Paul and Ringo"

fi
sysadmin@localhost:~/Documents$ bash hello.sh
George Harrison
```

```
sysadmin@localhost:~/Documents$ vi hello.sh
sysadmin@localhost:~/Documents$ cat hello.sh
NAMES=(Joe Jenny Sara Tony)

for N in ${NAMES[@]} ; do

  echo "My name is $N"

done



# loop on command output results

for f in $( ls prog.sh /etc/localtime ) ; do

  echo "File is: $f"

done
```

```
sysadmin@localhost:~/Documents$ bash hello.sh
My name is Joe
My name is Jenny
My name is Sara
My name is Tony
ls: cannot access prog.sh: No such file or directory
File is: /etc/localtime
```

```
sysadmin@localhost:~/Documents$ vi hello.sh
sysadmin@localhost:~/Documents$ cat hello.sh
# Prints out 0,1,2,3,4
COUNT=0
while [ $COUNT -ge 0 ]; do
  echo "Value of COUNT is: $COUNT"
  COUNT=$((COUNT+1))
  if [ $COUNT -ge 5 ] ; then
    break
  fi
done
# Prints out only odd numbers - 1,3,5,7,9
COUNT=0
while [ $COUNT -lt 10 ]; do
  COUNT=$((COUNT+1))

  # Check if COUNT is even
  if [ $(($COUNT % 2)) = 0 ] ; then
    continue
  fi
  echo $COUNT
done
```

```
sysadmin@localhost:~/Documents$ bash hello.sh
Value of COUNT is: 0
Value of COUNT is: 1
Value of COUNT is: 2
Value of COUNT is: 3
Value of COUNT is: 4
1
3
5
7
9
```

Lab Practical-4

Aim: Write a shell script that accept a file name starting and ending line numbers as arguments and display all the lines between given line no.

Software Used: Command Prompt.

Commands:

```
sysadmin@localhost:~/Documents$ cat animals.txt
1 retriever
2 badger
3 bat
4 wolf
5 eagle
```

```
sysadmin@localhost:~/Documents$ vi hello.sh
sysadmin@localhost:~/Documents$ cat hello.sh
echo "enter the filename"
read fname
echo "enter the starting line number"
read s
echo "enter the ending line number"
read n
sed -n $s,$n\p $fname | cat > newline
cat newline
sysadmin@localhost:~/Documents$ bash hello.sh
enter the filename
animals.txt
enter the starting line number
1
enter the ending line number
3
1 retriever
2 badger
3 bat
```

Lab Practical-5

Aim: Write a shell script that delete all lines containing a specified word.

Software Used: Command Prompt.

Commands:

```
sysadmin@localhost:~/Documents$ cat animals.txt
1 retriever
2 badger
3 bat
4 wolf
5 eagle
sysadmin@localhost:~/Documents$ vi hello.sh
sysadmin@localhost:~/Documents$ cat hello.sh
#! /bin/bash

echo "Enter the filename :"
read fname

echo "Enter the word :"
read word

sed -i '/'$word'/d' $fname

echo "lines containing given word are deleted"
```

```
sysadmin@localhost:~/Documents$ bash hello.sh
Enter the filename :
animals.txt
Enter the word :
bat
lines containing given word are deleted
sysadmin@localhost:~/Documents$ cat animals.txt
1 retriever
2 badger
4 wolf
5 eagle
```

Lab Practical-6

Aim: Write a shell script that displays a list of all the files in the current directory.

Software Used: Command Prompt.

Commands:

```
sysadmin@localhost:~$ vi hello.sh
sysadmin@localhost:~$ cat hello.sh
#! /bin/bash

echo "Enter the directory name"
read dir

if [ -d $dir ]
then
echo "list of files in the directory"
ls -l $dir

else
echo "Enter a proper directory name"
fi
```

```
sysadmin@localhost:~$ bash hello.sh
Enter the directory name
Documents
list of files in the directory
total 144
drwx------ 5 sysadmin sysadmin  4096 Dec 20  2017 School
drwx------ 2 sysadmin sysadmin  4096 Dec 20  2017 Work
-rw-r--r-- 1 sysadmin sysadmin    39 Dec 20  2017 adjectives.txt
-rw-r--r-- 1 sysadmin sysadmin    90 Dec 20  2017 alpha-first.txt
-rw-r--r-- 1 sysadmin sysadmin   106 Dec 20  2017 alpha-second.txt
-rw-r--r-- 1 sysadmin sysadmin   195 Dec 20  2017 alpha-third.txt
-rw-r--r-- 1 sysadmin sysadmin   390 Dec 20  2017 alpha.txt
-rw-r--r-- 1 sysadmin sysadmin    42 Dec 20  2017 animals.txt
-rw-r--r-- 1 sysadmin sysadmin    14 Dec 20  2017 food.txt
-rw-r--r-- 1 sysadmin sysadmin   647 Dec 20  2017 hello.sh
-rw-r--r-- 1 sysadmin sysadmin    67 Dec 20  2017 hidden.txt
-rw-r--r-- 1 sysadmin sysadmin    10 Dec 20  2017 letters.txt
-rw-r--r-- 1 sysadmin sysadmin    83 Dec 20  2017 linux.txt
-rw-r--r-- 1 sysadmin sysadmin 66540 Dec 20  2017 longfile.txt
-rw-r--r-- 1 sysadmin sysadmin   235 Dec 20  2017 newhome.txt
-rw-r--r-- 1 sysadmin sysadmin    10 Dec 20  2017 numbers.txt
```

Lab Practical-7

Aim: Simulation of Unix commands using C.

Software Used: Command Prompt.

Commands:

```
sysadmin@localhost:~/Documents$ nano list.c
```

```
  GNU nano 2.2.6                File: list.c

#include<stdio.h>
#include<stdlib.h>
#include<dirent.h>
int main() {
        char dirname[10];
        DIR*p;
        struct dirent *d;
        printf("Enter a directory name: ");
        scanf("%s",dirname);
        p=opendir(dirname);
        if(p==NULL)
        {
                perror("Cannot find directory");
                exit(-1);
        }
        while(d=readdir(p))
                printf("%s\n",d->d_name);
}

                            [ Read 18 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

```
sysadmin@localhost:~/Documents$ gcc list.c && ./a.out
Enter a directory name: School
.
Engineering
..
Math
Art
```

Lab Practical-8

Aim: Implement the following CPU Scheduling Algorithms.
i) FCFS ii) Shortest Job First

Language Used: C++

Commands:

FCFS- First Come First Serve process scheduling algorithm executes the processes in the order of their arrival in the ready queue. The process which arrives first is executed first.

```cpp
// C++ program for implementation of FCFS
// scheduling
#include<iostream>
using namespace std;

// Function to find the waiting time for all
// processes
void findWaitingTime(int processes[], int n,
                                                int bt[], int wt[])
{
  // waiting time for first process is 0
  wt[0] = 0;

  // calculating waiting time
  for (int i = 1; i < n ; i++ )
            wt[i] = bt[i-1] + wt[i-1] ;
}

// Function to calculate turn around time
void findTurnAroundTime( int processes[], int n,
                                int bt[], int wt[], int tat[])
{
  // calculating turnaround time by adding
  // bt[i] + wt[i]
  for (int i = 0; i < n ; i++)
            tat[i] = bt[i] + wt[i];
}

//Function to calculate average time
void findavgTime( int processes[], int n, int bt[])
{
  int wt[n], tat[n], total_wt = 0, total_tat = 0;

  //Function to find waiting time of all processes
  findWaitingTime(processes, n, bt, wt);

  //Function to find turn around time for all processes
  findTurnAroundTime(processes, n, bt, wt, tat);

  //Display processes along with all details
  cout << "Processes "<< " Burst time "
            << " Waiting time " << " Turn around time\n";

  // Calculate total waiting time and total turn
  // around time
  for (int i=0; i<n; i++)
  {
            total_wt = total_wt + wt[i];
            total_tat = total_tat + tat[i];
            cout << " " << i+1 << "\t\t" << bt[i] <<"\t "
                      << wt[i] <<"\t\t " << tat[i] <<endl;
  }

  cout << "Average waiting time = "
            << (float)total_wt / (float)n;
```

```
    cout << "\nAverage turn around time = "
              << (float)total_tat / (float)n;
}

// Driver code
int main()
{
    //process id's
    int processes[] = { 1, 2, 3};
    int n = sizeof processes / sizeof processes[0];

    //Burst time of all processes
    int burst_time[] = {10, 5, 8};

    findavgTime(processes, n, burst_time);
    return 0;
}

```

OUTPUT:

```
Processes  Burst time  Waiting time  Turn around time
 1              10         0              10
 2              5          10             15
 3              8          15             23
Average waiting time = 8.33333
Average turn around time = 16
```

SHORTEST JOB FIRST – The processes having the least burst times are executed first. This is a non-preemptive scheduling algorithm.

```
#include <bits/stdc++.h>
using namespace std;
//structure for every process
struct Process {
   int pid; // Process ID
   int bt; // Burst Time
   int art; // Arrival Time
};
void findTurnAroundTime(Process proc[], int n, int wt[], int tat[]) {
   for (int i = 0; i < n; i++)
   tat[i] = proc[i].bt + wt[i];
}
//waiting time of all process
void findWaitingTime(Process proc[], int n, int wt[]) {
   int rt[n];
   for (int i = 0; i < n; i++)
   rt[i] = proc[i].bt;
   int complete = 0, t = 0, minm = INT_MAX;
   int shortest = 0, finish_time;
   bool check = false;
   while (complete != n) {
      for (int j = 0; j < n; j++) {
         if ((proc[j].art <= t) && (rt[j] < minm) && rt[j] > 0) {
            minm = rt[j];
            shortest = j;
            check = true;
         }
      }
      if (check == false) {
         t++;
         continue;
      }
      // decrementing the remaining time
      rt[shortest]--;
      minm = rt[shortest];
      if (minm == 0)
         minm = INT_MAX;
         // If a process gets completely
         // executed
         if (rt[shortest] == 0) {
            complete++;
            check = false;
            finish_time = t + 1;
            // Calculate waiting time
            wt[shortest] = finish_time -
            proc[shortest].bt -
            proc[shortest].art;
            if (wt[shortest] < 0)
               wt[shortest] = 0;
         }
         // Increment time
         t++;
   }
}
// Function to calculate average time
void findavgTime(Process proc[], int n) {
   int wt[n], tat[n], total_wt = 0,
   total_tat = 0;
   // Function to find waiting time of all
   // processes
   findWaitingTime(proc, n, wt);
   // Function to find turn around time for
   // all processes
   findTurnAroundTime(proc, n, wt, tat);
   cout << "Processes " << " Burst time " << " Waiting time " << " Turn around time\n";
   for (int i = 0; i < n; i++) {
      total_wt = total_wt + wt[i];
```

```
68.       total_tat = total_tat + tat[i];
69.       cout << " " << proc[i].pid << "\t\t" << proc[i].bt << "\t\t " << wt[i] << "\t\t " << tat[i] << endl;
70.    }
71.    cout << "\nAverage waiting time = " << (float)total_wt / (float)n; cout << "\nAverage turn around time =
    " << (float)total_tat / (float)n;
72. }
73. // main function
74. int main() {
75.    Process proc[] = { { 1, 5, 1 }, { 2, 3, 1 }, { 3, 6, 2 }, { 4, 5, 3 } };
76.    int n = sizeof(proc) / sizeof(proc[0]);
77.    findavgTime(proc, n);
78.    return 0;
79. }
80.
```

OUTPUT:

```
Processes  Burst time  Waiting time  Turn around time
 1              5                3                8
 2              3                0                3
 3              6                12               18
 4              5                6                11

Average waiting time = 5.25
Average turn around time = 10
```

Lab Practical-9

Aim: Implement the following CPU Scheduling Algorithms.
i) Round Robin ii) priority based

Language Used: C++

Commands:

ROUND ROBIN: Each process is executed for a certain fixed time quantum and the execution of the processes is carried out in a cyclic manner. No process remains untouched in this scheduling algorithm. This a preemptive scheduling algorithm.

```
1.  // C++ program for implementation of RR scheduling
2.  #include<iostream>
3.  using namespace std;
4.
5.  // Function to find the waiting time for all
6.  // processes
7.  void findWaitingTime(int processes[], int n,
8.                          int bt[], int wt[], int quantum)
9.  {
10.   // Make a copy of burst times bt[] to store remaining
11.   // burst times.
12.   int rem_bt[n];
13.   for (int i = 0 ; i < n ; i++)
14.             rem_bt[i] = bt[i];
15.
16.   int t = 0; // Current time
17.
18.   // Keep traversing processes in round robin manner
19.   // until all of them are not done.
20.   while (1)
21.   {
22.             bool done = true;
23.
24.             // Traverse all processes one by one repeatedly
25.             for (int i = 0 ; i < n; i++)
26.             {
27.                       // If burst time of a process is greater than 0
28.                       // then only need to process further
29.                       if (rem_bt[i] > 0)
30.                       {
31.                                 done = false; // There is a pending process
32.
33.                                 if (rem_bt[i] > quantum)
34.                                 {
35.                                           // Increase the value of t i.e. shows
36.                                           // how much time a process has been processed
37.                                           t += quantum;
38.
39.                                           // Decrease the burst_time of current process
40.                                           // by quantum
41.                                           rem_bt[i] -= quantum;
42.                                 }
43.
44.                                 // If burst time is smaller than or equal to
45.                                 // quantum. Last cycle for this process
46.                                 else
47.                                 {
48.                                           // Increase the value of t i.e. shows
49.                                           // how much time a process has been processed
50.                                           t = t + rem_bt[i];
51.
52.                                           // Waiting time is current time minus time
53.                                           // used by this process
```

```
                                        wt[i] = t - bt[i];

                                        // As the process gets fully executed
                                        // make its remaining burst time = 0
                                        rem_bt[i] = 0;
                                    }
                                }
                        }

                        // If all processes are done
                        if (done == true)
                        break;
        }
}

// Function to calculate turn around time
void findTurnAroundTime(int processes[], int n,
                                                    int bt[], int wt[], int tat[])
{
        // calculating turnaround time by adding
        // bt[i] + wt[i]
        for (int i = 0; i < n ; i++)
                        tat[i] = bt[i] + wt[i];
}

// Function to calculate average time
void findavgTime(int processes[], int n, int bt[],
                                                                                int quantum)
{
        int wt[n], tat[n], total_wt = 0, total_tat = 0;

        // Function to find waiting time of all processes
        findWaitingTime(processes, n, bt, wt, quantum);

        // Function to find turn around time for all processes
        findTurnAroundTime(processes, n, bt, wt, tat);

        // Display processes along with all details
        cout << "Processes "<< " Burst time "
                        << " Waiting time " << " Turn around time\n";

        // Calculate total waiting time and total turn
        // around time
        for (int i=0; i<n; i++)
        {
                        total_wt = total_wt + wt[i];
                                total_tat = total_tat + tat[i];
                                cout << " " << i+1 << "\t\t" << bt[i] <<"\t "
                                        << wt[i] <<"\t\t " << tat[i] <<endl;
                        }

                        cout << "Average waiting time = "
                                << (float)total_wt / (float)n;
                        cout << "\nAverage turn around time = "
                                << (float)total_tat / (float)n;
        }

        // Driver code
        int main()
        {
                        // process id's
                        int processes[] = { 1, 2, 3};
                        int n = sizeof processes / sizeof processes[0];

                        // Burst time of all processes
                        int burst_time[] = {10, 5, 8};

                        // Time quantum
                        int quantum = 2;
                        findavgTime(processes, n, burst_time, quantum);
                        return 0;
        }
```

126.

OUTPUT:

```
Processes  Burst time  Waiting time  Turn around time
 1              10         13             23
 2              5          10             15
 3              8          13             21
Average waiting time = 12
Average turn around time = 19.6667
```

PRIORITY BASED: In priority-based CPU scheduling algorithm, each process is assigned a priority. Process with the highest priority is to be executed first and so on. Processes with same priority are executed on first come first served basis. Priority can be decided based on memory requirements, time requirements or any other resource requirements.

```
#include<bits/stdc++.h>
using namespace std;

struct Process
{
  int pid; // Process ID
  int bt; // CPU Burst time required
  int priority; // Priority of this process
};

// Function to sort the Process acc. to priority
bool comparison(Process a, Process b)
{
  return (a.priority > b.priority);
}

// Function to find the waiting time for all processes
void findWaitingTime(Process proc[], int n,
                                          int wt[])
{
  // waiting time for first process is 0
  wt[0] = 0;

  // calculating waiting time
  for (int i = 1; i < n ; i++ )
            wt[i] = proc[i-1].bt + wt[i-1] ;
}

// Function to calculate turn around time
void findTurnAroundTime( Process proc[], int n,int wt[], int tat[])
{
  // calculating turnaround time by adding bt[i] + wt[i]
  for (int i = 0; i < n ; i++)
            tat[i] = proc[i].bt + wt[i];
}

//Function to calculate average time
void findavgTime(Process proc[], int n)
{
  int wt[n], tat[n], total_wt = 0, total_tat = 0;

  //Function to find waiting time of all processes
  findWaitingTime(proc, n, wt);

  //Function to find turn around time for all processes
  findTurnAroundTime(proc, n, wt, tat);

  //Display processes along with all details
  cout << "\nProcesses "<< " Burst time "
            << " Waiting time " << " Turn around time\n";

  // Calculate total waiting time and total turn around time
  for (int i=0; i<n; i++)
  {
            total_wt = total_wt + wt[i];
            total_tat = total_tat + tat[i];
            cout << " " << proc[i].pid << "\t\t"
                      << proc[i].bt << "\t " << wt[i]
                      << "\t\t " << tat[i] <<endl;
  }

  cout << "\nAverage waiting time = "
            << (float)total_wt / (float)n;
  cout << "\nAverage turn around time = "
```

```
            << (float)total_tat / (float)n;
}

void priorityScheduling(Process proc[], int n)
{
  // Sort processes by priority
  sort(proc, proc + n, comparison);

  cout<< "Order in which processes gets executed \n";
  for (int i = 0 ; i < n; i++)
            cout << proc[i].pid <<" " ;

  findavgTime(proc, n);
}

// Driver code
int main()
{
  Process proc[] = {{1, 10, 2}, {2, 5, 0}, {3, 8, 1}};
  int n = sizeof proc / sizeof proc[0];
  priorityScheduling(proc, n);
  return 0;
}
```

OUTPUT:

```
Order in which processes gets executed
1 3 2
Processes  Burst time  Waiting time  Turn around time
 1              10         0              10
 3              8          10             18
 2              5          18             23

Average waiting time = 9.33333
Average turn around time = 17
```

Lab Practical-10

Aim: Write a program to implement banker's algorithm.

Language Used: C++

Commands:

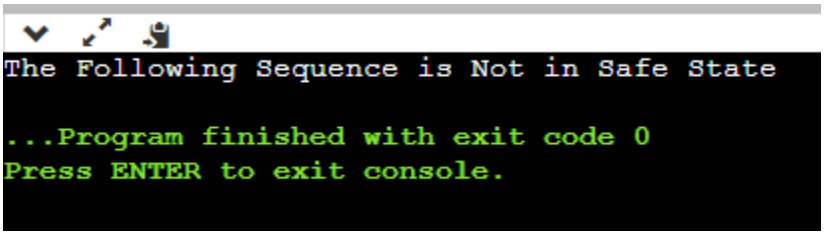
Banker's algorithm is a deadlock avoidance algorithm for resources with more than one instance. It utilizes the static information about the allocation and availability of the resources along with the maximum need of all the processes. If at any instance, the need of the processes exceeds the availability of the resources, the execution of the process becomes stunted, there is absence of a safe sequence of execution of the processes and thus the state is said to be unsafe and deadlock is vulnerable to occur.

```
1. /*This code is for executing Banker's Algorithm*/
2.
3. #include <iostream>
4. using namespace std;
5.
6. int main()
7. {
8.   // P0, P1, P2, P3, P4 are the Process names here
9.
10.    int iProcess, iResource, iLoop, jLoop, kLoop;
11.    iProcess = 5;      // Number of Processes
12.    iResource = 3;      // Number of Resources
13.
14.    // This Matrix is for Allocating the resources
15.
16.    int alloc[5][3] = { { 0, 1, 0 },    // P0
17.                                       { 2, 0, 0 },    // P1
18.                                       { 3, 0, 2 },    // P2
19.                                       { 2, 1, 1 },    // P3
20.                                       { 0, 0, 2 } };  // P4
21.
22.    // This Matrix is for Maximum Resources
23.
24.    int max[5][3] = {   { 7, 5, 3 },    // P0
25.                                     { 3, 2, 2 },    // P1
26.                                     { 9, 0, 2 },    // P2
27.                                     { 2, 2, 2 },    // P3
28.                                     { 4, 3, 3 } };  // P4
29.
30.    // This is for showing the number of available resorces at an instance
31.    int avail[3] = { 3, 3, 2 };
32.
33.    int f[iResource], ans[iResource], ind = 0;
34.    for (kLoop = 0; kLoop < iResource; kLoop++)
35.    {
36.      f[kLoop] = 0;
37.    }
38.
39.int need[iResource][iProcess];
40.
41.for (iLoop = 0; iLoop < iResource; iLoop++)
42.{
43.  for (jLoop = 0; jLoop < iProcess; jLoop++)
44.  {
```

```
        // Need = Max - Allocated
        need[iLoop][jLoop] = max[iLoop][jLoop] - alloc[iLoop][jLoop];
  }
}

int y = 0;
for (kLoop = 0; kLoop < 5; kLoop++)
{
  for (iLoop = 0; iLoop < iResource; iLoop++)
  {
      if (f[iLoop] == 0)
      {
              int flag = 0;
              for (jLoop = 0; jLoop < iProcess; jLoop++)
          {
              if ( need[iLoop][jLoop] > avail[jLoop] )
              {
                      flag = 1;
                      break;
              }
          }

          if (flag == 0)
          {
              ans[ind++] = iLoop;
              for (y = 0; y < iProcess; y++)
              {
                //  Available = Available + Allocated
                      avail[y] += alloc[iLoop][y];
              }

            f[iLoop] = 1;
          }
  }
}

int flag = 1;

// For Checking whether the following Sequence is in Safe State or Not.

for(int iLoop = 0; iLoop < iResource; iLoop++)
{
  if(f[iLoop]==0)
  {
        flag = 0;
        cout << "The Following Sequence is Not in Safe State";
        break;
    }
}

if(flag==1)
{
    cout << "The Following Sequence is in Safe State";
  cout << "\n\n---Safe Sequence---" << endl;

          for (iLoop = 0; iLoop < iResource - 1; iLoop++)
          {
              cout << " P" << ans[iLoop] << " --->";
          }

          cout << " P" << ans[iResource - 1] <<endl;
```

```
}
}
        return (0);
}
}
```

OUTPUT:

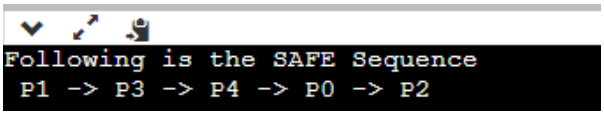


For Executing the Same code for Safe State -

```
// Banker's Algorithm
#include <iostream>
using namespace std;

int main()
{
  // P0, P1, P2, P3, P4 are the Process names here

int n, m, i, j, k;
n = 5; // Number of processes
m = 3; // Number of resources
int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix
                                  { 2, 0, 0 }, // P1
                                  { 3, 0, 2 }, // P2
                                  { 2, 1, 1 }, // P3
                                  { 0, 0, 2 } }; // P4

int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix
                           { 3, 2, 2 }, // P1
                           { 9, 0, 2 }, // P2
                           { 2, 2, 2 }, // P3
                           { 4, 3, 3 } }; // P4

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) {
```

```
39.
40.           int flag = 0;
41.           for (j = 0; j < m; j++) {
42.           if (need[i][j] > avail[j]){
43.                   flag = 1;
44.                   break;
45.           }
46.           }
47.
48.           if (flag == 0) {
49.           ans[ind++] = i;
50.           for (y = 0; y < m; y++)
51.                   avail[y] += alloc[i][y];
52.           f[i] = 1;
53.           }
54.   }
55.   }
56. }
57.
58. int flag = 1;
59.
60. // To check if sequence is safe or not
61. for(int i = 0;i<n;i++)
62. {
63.           if(f[i]==0)
64.   {
65.           flag = 0;
66.           cout << "The given sequence is not safe";
67.           break;
68.   }
69. }
70.
71. if(flag==1)
72. {
73.   cout << "Following is the SAFE Sequence" << endl;
74.   for (i = 0; i < n - 1; i++)
75.           cout << " P" << ans[i] << " ->";
76.   cout << " P" << ans[n - 1] <<endl;
77. }
78.
79.   return (0);
80. }
```

OUTPUT:



Lab Practical-11

Aim: Write a program to implement paging algorithm.

Language Used: C

Commands:

In a computer operating system that uses paging for virtual memory management, page replacement algorithms decide which memory pages to page out, sometimes called swap out, or write to disk, when a page of memory needs to be allocated.

```c
// C program for FIFO page replacement algorithm
#include <stdio.h>
int main()
{
    int incomingStream[] = {4, 1, 2, 4, 5};
    int pageFaults = 0;
    int frames = 3;
    int m, n, s, pages;

    pages = sizeof(incomingStream)/sizeof(incomingStream[0]);

    printf("Incoming \t Frame 1 \t Frame 2 \t Frame 3");
    int temp[frames];
    for(m = 0; m < frames; m++)
    {   temp[m] = -1;
    }

    for(m = 0; m < pages; m++)
    {   s = 0;

        for(n = 0; n < frames; n++)
        {   if(incomingStream[m] == temp[n])
            {
                s++;
                pageFaults--;
            }
        }
        pageFaults++;

        if((pageFaults <= frames) && (s == 0))
        {   temp[m] = incomingStream[m];
        }
        else if(s == 0)
        {   temp[(pageFaults - 1) % frames] = incomingStream[m];
        }

        printf("\n");
        printf("%d\t\t\t",incomingStream[m]);
        for(n = 0; n < frames; n++)
        {   if(temp[n] != -1)
                printf(" %d\t\t\t", temp[n]);
            else
                printf(" - \t\t\t");
        }
    }

    printf("\nTotal Page Faults:\t%d\n", pageFaults);
```

```
48.     return 0;
49. }
```

OUTPUT:

```
Incoming          Frame 1          Frame 2          Frame 3
4                       4                          -                          -
1                       4                          1                          -
2                       4                          1                          2
4                       4                          1                          2
5                       5                          1                          2
Total Page Faults:      4
```

OPEN ENDED PROJECT

Aim: Write a script which will shows all running process on your Linux system.

Software Used: Command Prompt.

Theory: A process is nothing but tasks within the Linux operating system. You can list running processes using the ps command (ps means process status). Both Linux and UNIX support the ps command to display information about all running process. The ps command gives a snapshot of the current processes. If you want a repetitive update of this status, use top command.

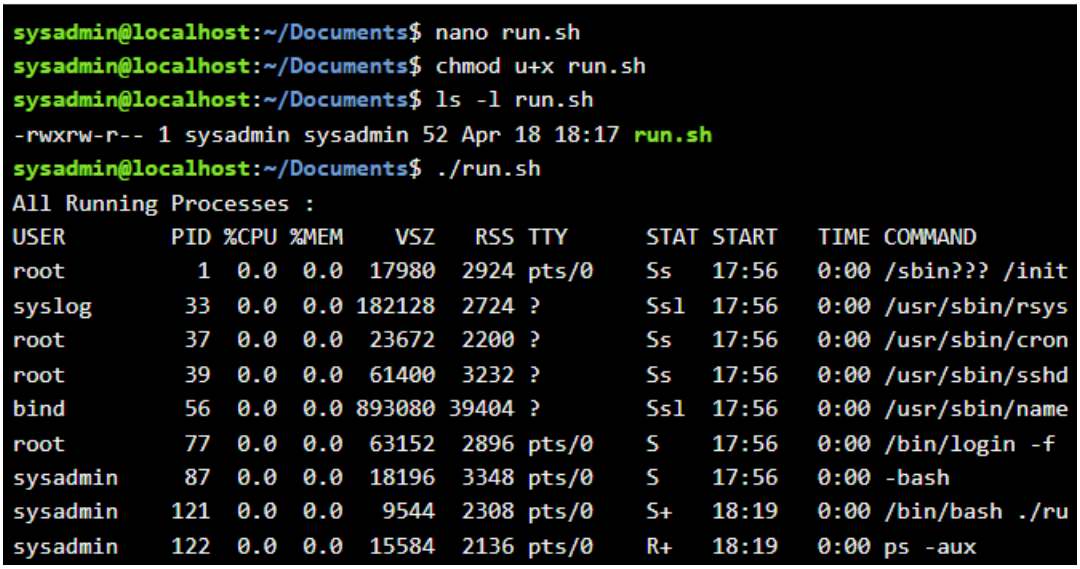
ps - display the processes running in the current shell.

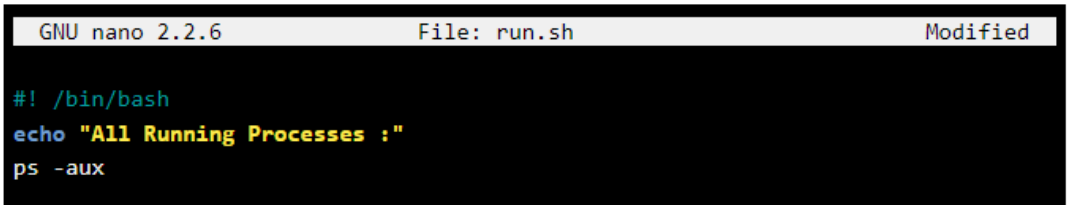
ps -ef or ps -aux (display all the currently running processes)

top - It shows the linux processes. It provides a dynamic real-time view of the running system.

Commands:

```
#! /bin/bash
echo "All Running Processes :"
ps -aux
```





OPEN ENDED PROJECT

Aim: WAP to generate maximum number of child process in your system and with the help of program explain what Zombie process are.

Software Used: Command Prompt.

Theory: Zombie state: When a process is created in UNIX using fork() system call, the address space of the Parent process is replicated. If the parent process calls wait() system call, then the execution of the parent is suspended until the child is terminated. At the termination of the child, a 'SIGCHLD' signal is generated which is delivered to the parent by the kernel. Parent, on receipt of 'SIGCHLD' reads the status of the child from the process table. Even though the child is terminated, there is an entry in the process table corresponding to the child where the status is stored. When the parent collects the status, this entry is deleted. Thus, all the traces of the child process are removed from the system. If the parent decides not to wait for the child's termination and executes its subsequent task, then at the termination of the child, the exit status is not read. Hence, there remains an entry in the process table even after the termination of the child. This state of the child process is known as the Zombie state.

Commands:

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

int main()
{
    int i;
    int pid = fork();

    if (pid == 0)
    {
        for (i=0; i<20; i++)
            printf("I am Child\n");
    }
    else
    {
        printf("I am Parent\n");
        while(1);
    }
}
```

```
sysadmin@localhost:~/Documents$ nano zombie1.c
sysadmin@localhost:~/Documents$ chmod u+x zombie1.c
sysadmin@localhost:~/Documents$ cc zombie1.c
sysadmin@localhost:~/Documents$ ./a.out
```

```
sysadmin@localhost:~/Documents$ ./a.out
I am Parent
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
I am Child
```

```
 GNU nano 2.2.6                File: zombie1.c                          Modified

#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
#include<sys/types.h>
int main()
{
    int i;
    int pid = fork();
    if (pid == 0)
    {
        for (i=0; i<20; i++)
            printf("I am Child\n");
    }
    else
    {
        printf("I am Parent\n");
        while(1);
    }
}
```