INTRODUCTION TO LINUX

EXPERIMENT NO:1
DATE:09/05/2022

AIM:
To familiarise with various Linux concepts:

- Linux and various UNIX variants
- Linux architecture diagram
- Linux directory structure
- Shells, bash shell

Theory

Linux is the family of open source unix-like operating systems based on the linux kernel,an operating system kernel which was first released on september 17 1991 by Linus Torvalds.Linux Kernel is open source software.The Kernel is the core part of the operating system that interacts directly with the hardware, in other words the kernel provides basic functionality for all other parts of the operating system.

UNIX Variants
There is no single operating UNIX operating system. Instead, there is a large collection of UNIX variants. All these variants share a large number of features.The most widely used UNIX variants, including Linux, Solaris, HP-UX, Mac OS X, and AIX.

Linux is an extremely popular variant of the UNIX System. Among the reasons for this popularity is that it can be used free of charge, as well as the depth and breadth of its capabilities and the large amount of software that runs on Linux.

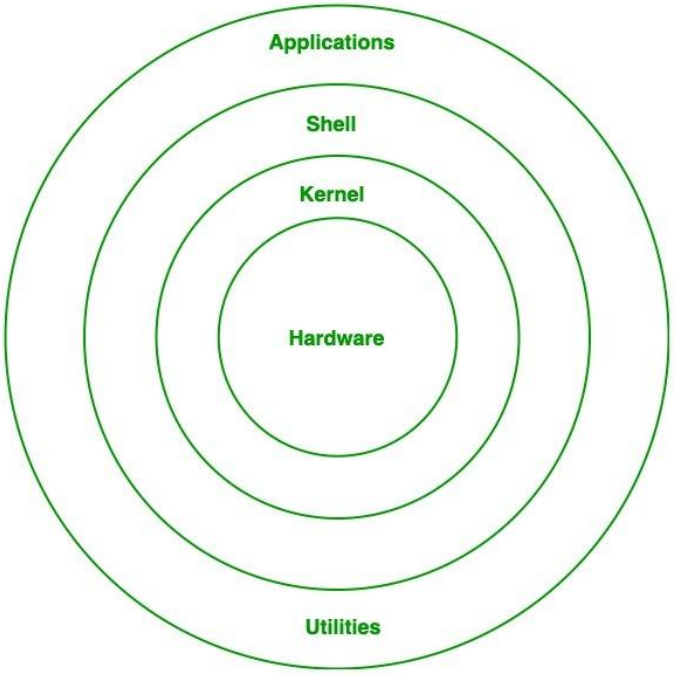
Solaris : The original operating system of Sun Microsystems was called the SunOS. It was based on UNIX System V Release 2 and 4.3BSD. SunSoft's first version of UNIX, Solaris 1.0, was an enhanced version of the SunOS.

MAC OS X: The Mac OS, the operating system developed by Apple Computer for its Macintosh computers, was first developed in 1984.In particular, the Mac OS had an entirely graphical user interface with no command-line interface.

AIX: The UNIX system version for IBM machines is called AIX and is based on System V version 3 and BSD 4.3

HP-UX:It is the version of UNIX made by the computer manufacturer Hewlett-Packard, based on UNIX System V version 2

Linux Architecture Diagram



Kernel: Kernel is the core of the Linux based operating system. It virtualizes the common hardware resources of the computer to provide each process with its virtual resources. This makes the process seem as if it is the sole process running on the machine. The kernel is also responsible for preventing and mitigating conflicts between different processes. Different types of the kernel are: Monolithic Kernel, Hybrid Kernels, Exo Kernels, Micro Kernels.

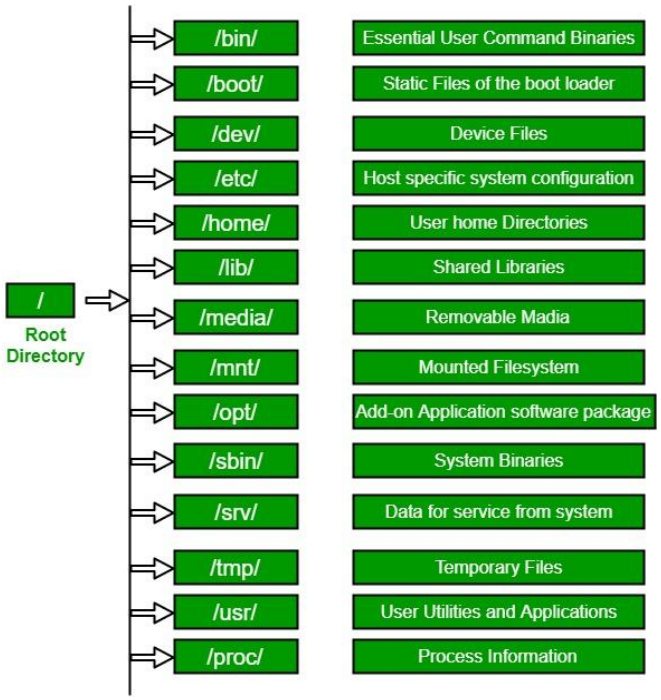
System Library: The special types of functions that are used to implement the functionality of the operating system.

Shell: It is an interface to the kernel which hides the complexity of the kernel's functions from the users. It takes commands from the user and executes the kernel's functions.

Hardware Layer: This layer consists of all peripheral devices like RAM/ HDD/ CPU etc.

System Utility: It provides the functionalities of an operating system to the user.

LINUX Directory Structure



The Linux Foundation maintains a Filesystem Hierarchy Standard (FHS). This FHS defines the directory structure and the content/purpose of the directories in Linux distributions.

/ – The root directory : Everything, all the files and directories, in Linux are located under 'root' represented by '/'.

/bin – Binaries: The '/bin' directory contains the executable files of many basic shell commands like ls, cp, cd etc. Mostly the programs are in binary format here and accessible by all the users in the Linux system.

/dev – Device files: This directory only contains special files, including those relating to the devices. These are virtual files, not physically on the disk.

/etc – Configuration files: The /etc directory contains the core configuration files of the system, used primarily by the administrator and services, such as the password file and networking files

/home – User personal data: The home directory contains the user data and user-specific configuration files.

/lib – Shared libraries:This holds the libraries needed by the binaries in /bin and /sbin directories. /sbin - Programs for use by the system and the system administrator.

/tmp - Temporary space for use by the system, cleaned upon reboot.

/var - Storage for all variable files and temporary files created by users, such as log files, the mail queue, space for temporary storage of files downloaded from the Internet etc.

/boot - The '/boot' directory contains the files of the kernel and boot image, in addition to LILO and Grub.

/proc - The '/proc' directory contains the information about currently running processes and kernel parameters.

/opt - Typically contains extra and third party softwares.

/mnt - Standard mount point for external file systems, e.g. a CD-ROM or a digital camera

Types of Shells in LINUX

The different Types of Shells in Linux can offer various capabilities, but at their core, they're basically implementing ideas that were developed decades ago.

1. Bourne-Again Shell

Bash stands for Bourne Again Shell and it is the default shell on many Linux distributions today.

2. TENEX C Shell

Tcsh is an enhanced C shell, it can be used as an interactive login shell and shell script command processor. Tcsh has the following features:

- (a) C like syntax
- (b) Command-line editor
- (c) Programmable word and filename completion
- (d) Spelling correction
- (e) Job control

3. Korn Shell

Ksh stands for Korn shell and was designed and developed by David G. Korn. It is a complete, powerful, high-level programming language and also an interactive command language just like many other Unix/GNU Linux shells.

The Korn shell includes features like associative arrays, floating point arithmetic, job control, command aliasing, command history, supports POSIX standards, backward compatibility with bash.

4. Z Shell

Zsh is designed to be interactive and it incorporates many features of other Unix/GNU Linux shells such as bash, tcsh, and ksh.

It is also a powerful scripting language just like the other shells available. Though it has some unique features that include: Filename generation, Startup files, Login/Logout watching, Closing comments, Concept index, Variable index, Functions index, Key index.

5. Scheme Shell

The Scheme shell (scsh) is an exotic shell that offers a scripting environment using Scheme, which is a derivative of the Lisp language. The Pyshell is an attempt to create a similar script that uses the Python language.

BASH

Bash is the shell, or command language interpreter, for the GNU operating system. The name is an acronym for the ‘Bourne-Again SHell’, a pun on Stephen Bourne, the author of the direct ancestor of the current Unix shell sh, which appeared in the Seventh Edition Bell Labs Research version of Unix.

Bash is largely compatible with sh and incorporates useful features from the Korn shell ksh and the C shell csh. It is intended to be a conformant implementation of the IEEE POSIX Shell and Tools portion of the IEEE POSIX specification (IEEE Standard 1003.1). It offers functional improvements over sh for both interactive and programming use.

While the GNU operating system provides other shells, including a version of csh, Bash is the default shell. Like other GNU software, Bash is quite portable. It currently runs on nearly every version of

Unix and a few other operating systems - independently-supported ports exist for MS-DOS, OS/2, and Windows platforms.

It offers practical improvements over sh for programming and interactive use which includes:

(a) Command line editing
(b) Job Control
(c) Unlimited size command history
(d) Shell Functions and Aliases
(e) Unlimited size Indexed arrays
(f) Integer arithmetic in any base from two to sixty-four

RESULT

Familiarised with linux and its basic concepts.

FAMILIARISATION OF BASIC LINUX COMMANDS

EXPERIMENT NO:2
DATE:17/05/2022

AIM:

To familiarise basic linux commands- ls, mkdir, rmdir, cd, touch, rm, pwd, cp, mv.

DESCRIPTION

ls - To list the contents of a directory. To see the content of other directories, type ls and then the directory's path.
Some options available with ls command are:

ls -R : will list all the files in the sub-directories as well
ls -a: will show the hidden files
ls -al: will list the files and directories with detailed information like the permissions, size, owner, etc.

mkdir - To make a new directory. Syntax : 'mkdir directory_name'.

rmdir - To remove an empty directory. Syntax : 'rmdir directory_name'

cd - To change the current working directory. Some shortcuts available with cd command are: **cd .. :** (with two dots) to move one directory up
cd : to go straight to the home folder
cd- : (with a hyphen) to move to your previous directory

touch - To create one or more files through the Linux command line.

rm - To delete a file. Syntax is 'rm filename.ext'.
To delete a non.empty directory with its contents, we can use 'rm -r directory_name'

pwd - To find out the path of the current working directory (folder) you're in.It will return an absolute (full) path.

cp - To copy files from the current directory to a different directory. Syntax is 'cp filename.ext destination_directory'.

mv - To move files from one directory to another, the syntax is 'mv filename.ext destination_directory'. It is also used to rename files and for that the syntax is 'mv oldname.ext newname.ext'

OUTPUT

```
user@ubuntu:~$ cd /
user@ubuntu:/$ ls
bin   dev  home  lib    lib64   media  opt   root  sbin  srv  tmp  var
boot  etc  init  lib32  libx32  mnt    proc  run   snap  sys  usr
user@ubuntu:/$ ls -a
.   bin   dev  home  lib    lib64   media  opt   root  sbin  srv  tmp  var
..  boot  etc  init  lib32  libx32  mnt    proc  run   snap  sys  usr
user@ubuntu:/$ ls -al
total 620
drwxr-xr-x  1 root root   4096 Aug  2 20:38 .
drwxr-xr-x  1 root root   4096 Aug  2 20:38 ..
lrwxrwxrwx  1 root root      7 Mar 25 03:10 bin -> usr/bin
drwxr-xr-x  1 root root   4096 Mar 25 03:17 boot
drwxr-xr-x  1 root root   4096 Aug  2 20:38 dev
drwxr-xr-x  1 root root   4096 Aug  2 20:39 etc
drwxr-xr-x  1 root root   4096 Aug  2 20:38 home
-rwxr-xr-x  1 root root 632096 Aug  2 20:35 init
lrwxrwxrwx  1 root root      7 Mar 25 03:10 lib -> usr/lib
lrwxrwxrwx  1 root root      9 Mar 25 03:10 lib32 -> usr/lib32
lrwxrwxrwx  1 root root      9 Mar 25 03:10 lib64 -> usr/lib64
lrwxrwxrwx  1 root root     10 Mar 25 03:10 libx32 -> usr/libx32
drwxr-xr-x  1 root root   4096 Mar 25 03:10 media
drwxr-xr-x  1 root root   4096 Aug  2 20:38 mnt
drwxr-xr-x  1 root root   4096 Mar 25 03:10 opt
dr-xr-xr-x  9 root root      0 Aug  2 20:38 proc
drwx------  1 root root   4096 Mar 25 03:12 root
drwxr-xr-x  1 root root   4096 Aug  2 20:46 run
lrwxrwxrwx  1 root root      8 Mar 25 03:10 sbin -> usr/sbin
drwxr-xr-x  1 root root   4096 Mar 25 03:12 snap
drwxr-xr-x  1 root root   4096 Mar 25 03:10 srv
dr-xr-xr-x 12 root root      0 Aug  2 20:38 sys
drwxrwxrwt  1 root root   4096 Aug  2 20:39 tmp
drwxr-xr-x  1 root root   4096 Mar 25 03:11 usr
drwxr-xr-x  1 root root   4096 Mar 25 03:12 var
user@ubuntu:/$
```

```
user@ubuntu:~$ ls -R
.:
Java  OSLAB

./Java:
Calculator.java

./OSLAB:
test.c
user@ubuntu:~$
```

```
user@ubuntu:~$ cat text.txt
abcdefg
user@ubuntu:~$ cat file.txt
cat: file.txt: No such file or directory
user@ubuntu:~$ cp text.txt file.txt
user@ubuntu:~$ cat file.txt
abcdefg
```

```
user@ubuntu:~$ ls
java  oslab
user@ubuntu:~$ mkdir dslab
user@ubuntu:~$ mkdir dslab/c
user@ubuntu:~$ ls dslab
c
user@ubuntu:~$ mkdir dslab/c/a/b
mkdir: cannot create directory ‘dslab/c/a/b’: No such file or directory
user@ubuntu:~$ mkdir -p dslab/c/a/b
user@ubuntu:~$ ls -R
.:
dslab  java  oslab

./dslab:
c

./dslab/c:
a

./dslab/c/a:
b

./dslab/c/a/b:

./java:

./oslab:
user@ubuntu:~$
```

```
user@ubuntu:~$ ls -R
.:
dslab  java  oslab

./dslab:
c

./dslab/c:
a

./dslab/c/a:
b

./dslab/c/a/b:

./java:
test.c

./oslab:
user@ubuntu:~$ rmdir java
rmdir: failed to remove 'java': Directory not empty
user@ubuntu:~$ rmdir oslab
user@ubuntu:~$ rmdir -p dslab/c/a/b
user@ubuntu:~$ ls
java
user@ubuntu:~$
```

```
user@LAPTOP-BPURTOLO:~$ ls
Docs  Downloads  Music  Pictures  calc.sh  err.txt  exp.txt  exp1.txt  exp2.txt  exp5.txt  script.sh  test.txt  text.txt
user@LAPTOP-BPURTOLO:~$ chmod 777 exp2.txt
user@LAPTOP-BPURTOLO:~$ chmod 731 exp1.txt
user@LAPTOP-BPURTOLO:~$ chmod u="rwx" exp1.txt
user@LAPTOP-BPURTOLO:~$ ls -l
total 4
drwxr-xr-x 1 user user 4096 Aug  2 21:41 Docs
drwxr-xr-x 1 user user 4096 Aug  2 21:24 Downloads
drwxr-xr-x 1 user user 4096 Aug  2 21:24 Music
drwxr-xr-x 1 user user 4096 Aug  2 21:25 Pictures
-rwxr-xr-x 1 user user  698 Aug  3 09:43 calc.sh
-rw-r--r-- 1 user user  177 Aug  2 23:33 err.txt
-rw-r--r-- 1 user user   70 Aug  3 00:19 exp.txt
-rwx-wx--x 1 user user   16 Aug  2 23:30 exp1.txt
-rwxrwxrwx 1 user user   18 Aug  2 22:53 exp2.txt
-rw-r--r-- 1 user user   20 Aug  2 23:57 exp5.txt
-rwxr-xr-x 1 user user  352 Aug  3 00:57 script.sh
-rw-r--r-- 1 user user   30 Aug  2 23:54 test.txt
-rw-r--r-- 1 root test    8 Aug  2 21:31 text.txt
user@LAPTOP-BPURTOLO:~$
```

```
user@ubuntu:/$ cd
user@ubuntu:~$ cd ..
user@ubuntu:/home$ cd user/Downloads/
user@ubuntu:~/Downloads$ cd /home/user/Pictures/
user@ubuntu:~/Pictures$ cd /
user@ubuntu:/$ cd ~
user@ubuntu:~$ touch file.txt
user@ubuntu:~$ ls
Documents  Downloads  Music  Pictures  file.txt
user@ubuntu:~$ touch Documents/file.txt
user@ubuntu:~$ ls Documents/
file.txt
user@ubuntu:~$ rm file.txt
user@ubuntu:~$ rm -i Documents/file.txt
rm: remove regular empty file 'Documents/file.txt'? y
user@ubuntu:~$ pwd
/home/user
user@ubuntu:~$ cd /
user@ubuntu:/$ pwd
/
```

```
user@ubuntu:~$ ls
Documents  Downloads  Music  Pictures  file.txt  text.txt
user@ubuntu:~$ mv file.txt Documents/
user@ubuntu:~$ ls Documents/
file.txt
user@ubuntu:~$ mv Documents Docs
user@ubuntu:~$ ls
Docs  Downloads  Music  Pictures  text.txt
user@ubuntu:~$
```

RESULT

Familiarised with basic linux commands.

LINUX COMMANDS FOR OPERATION

EXPERIMENT NO:3
DATE:17/05/2022

AIM : To familiarise the linux commands for operation

DESCRIPTION

a).Changing ownership/permissions of files/links/directory

I) chmod:

PERMISSION	chmod option
read	r or 4
write	w or 2
execute	x or 1

i) read restricts or allows viewing the directories contents, i.e. ls command
ii) write restricts or allows creating new files or deleting files in the directory.
iii) execute restricts or allows changing into the directory, i.e. cd command

Option	Definition
r	Read
+	Add permission
-	Remove permission
=	Set permission

II) chmod using letters

Option	Definition
u	Owner
g	Group
o	other
a	all
x	Execute
w	write

Option	Definition
1	execute
2	write
4	read

III) chmod using numbers

Option	Definition
#--	Owner
-#-	Group
--#	Other

IV)chown , sudo , adduser , passwd

chown : A file's owner can be changed using the chown command.
Eg : to change the foobar file's owner to tux: sudo chown tux foobar
To change the foobar file's group to penguins, use chgrp or chown with special syntax: sudo chgrp penguins foobar or sudo chown :penguins foobar

adduser : To add a user account - sudo adduser username
To add a user to a group - sudo adduser username groupname

passwd : To enable the root account, give it a password -sudo passwd
To disable the root account - sudo passwd -l root
To temporarily lock or unlock a user account - sudo passwd -l username
sudo passwd -u username

b) Redirection

I) cat (concatenate) : It reads data from the file and gives their content as output. It helps us to create, view, concatenate files

$cat filename - It will show content of given filename

$cat file1 file2 - This will show the content of file1 and file2

$cat -n filename - It will show content with line number

$ cat >newfile - Will create and a file named newfile

$cat [filename-whose-contents-is-to-be-copied] > [destination-filename]

-The content will be copied in destination file

$cat -s filename - Will suppress repeated empty lines in output

$cat file1 >> file2 - Will append the contents of one file to the end of another file

$cat >> file.txt - Will append the text to the end of the file

II) input redirection

cat < file.txt- cat command will take the input from “file.txt” and print it to the terminal screen

III)Output Redirection

cat > file.txt - whatever you will write after running this command, all will be redirected and copied to the “file.txt”

IV)error redirection

Error redirection is transferring the errors generated by some false commands to a file rather than STDOUT.

$ command>error.txt - Using “2>” re-directs the error output to a file named “error.txt” and nothing is displayed on STDOUT.

c) Pipes

A pipe is a form of redirection that is used to send the output of one command to another command for further processing. The pipe character '|' is used for this

Pipe is used to combine two or more commands, and in this, the output of one command acts as input to another command, and this command's output may act as input to the next command and so on. The command line programs that do the further processing are referred to as filters.

Pipes are unidirectional i.e data flows from left to right through the pipeline.

$ ls -l | more - The more command takes the output of $ ls -l as its input. The pipe acts as a container which takes the output of ls -l and gives it to more as input.

d) Filters

I)head : print the top N number of data of the given input

i) **-n num**: Prints the first 'num' lines - eg $ head -n 5 state.txt

ii) **-c num**: Prints the first 'num' bytes from the file specified- eg $ head -c 6 state.txt

iii) **-q**: It is used if more than 1 file is given. Because of this command,data from each file is not preceded by its filename - eg $ head -q state.txt capital.txt

iv**) -v**: By using this option, data from the specified file is always preceded by its file name – eg $ head -v state.txt

head with pipeline : $ ls -t | head -n 3 (display three most recently used file)

II)tail : print the last N number of data of the given input

i)**-n num**: Prints the last 'num' – eg $ tail -n 3 state.txt

ii) **tail +n filename**:data will start printing from line number 'n' till the end of the file specified- eg $ tail +25 state.txt

iii) **-c num**: Prints the last 'num' bytes from the file specified

+num display all the data after skipping num bytes from starting of the specified file- eg $ tail -c 6 state.txt

-num it display the last num bytes from the file specified- eg $ tail -c -6 state.txt

iv) **-q**: It is used if more than 1 file is given- eg $ tail -q state.txt capital.txt

v) **-v**: By using this option, data from the specified file is always preceded by its file name- eg $ tail -v state.txt

tail with pipeline: $ tail -n 7 state.txt | sort –r(output of the tail command is given as input to the sort command with -r option to sort the last 7 state names coming from file state.txt in the reverse order)

III)wc(word count) : It is used to find out the number of lines, word count, byte and characters count in the files. By default it displays four-columnar output.

First column shows number of lines present in a file specified, second column shows number of words present in the file, third column shows number of characters present in file and fourth column is the file name which is given as argument.

Passing only one file name in the argument **- $ wc state.txt**

Passing more than one file name in the argument - **$ wc state.txt capital.txt**

(When more than file name is specified in argument then command will display four-columnar output for all individual files plus one extra row displaying total number of lines, words and characters of all the files specified in argument, followed by keyword total.)

i)**-l**: This option prints the number of lines present in a file. With this option wc command displays two-columnar output, 1st column shows number of lines present in a file and 2nd itself represents the file name.

With one file name - **$ wc -l state.txt**

With more than one file name - **$ wc -l state.txt capital.txt**

ii) **-w**: This option prints the number of words present in a file. With this option wc command displays two-columnar output, 1st column shows number of words present in a file and 2nd is the file name.

With one file name - **$ wc -w state.txt**

With more than one file name - **$ wc -w state.txt capital.txt**

iii) **-c**: This option displays the count of bytes present in a file. With this option it displays two-column output, 1st column shows number of bytes present in a file and 2nd is the file name.

With one file name - **$ wc -c state.txt**

With more than one file name - **$ wc -c state.txt capital.txt**

iv) **-m**: Using -m option ‘wc’ command displays count of characters from a file.

With one file name - **$ wc -m state.txt**

With more than one file name - **$ wc -m state.txt capital.txt**

IV) sort : used to sort a file - **$ sort filename.txt**

i)**-r Option**: Sorting In Reverse Order - $ sort -r inputfile.txt

ii) **-n Option** : To sort a file numerically - $ sort -n filename.txt

iii) **-nr** option : To sort a file with numeric data in reverse order - $ sort -nr filename.txt

iv) **-k Option** : sorting a table on the basis of any column number by using -k option - $ sort -k filename.txt

v) **-u option** : To sort and remove duplicates - $ sort -u filename.txt

V) tr(translate): used for translating or deleting characters

i)to convert lower case to upper case : **$cat greekfile | tr “[a-z]” “[A-Z]”**

ii) **-d option** - to delete specified characters : **$ echo "Welcome" | tr -d 'w'**

iii) **-s** : replaces repeated characters listed in the set1 with single occurrence

VI)sed(stream editor):

i)Replacing or substituting string : **$sed 's/unix/linux/' file.txt**

replaces the word “unix” with “linux” in the file(replaces only the first occurrence of the pattern in each line)

ii)Replacing the nth occurrence of a pattern in a line: **$sed 's/unix/linux/2' file.txt**

iii)Replacing all the occurrence of the pattern in a line : **$sed 's/unix/linux/g' file.txt**

VII)uniq

i)to delete repeated lines - **$uniq file.txt**(to delete all the repeated lines file should be sorted)

ii) **-c option** : It tells the number of times a line was repeated - $uniq -c file.txt

iii**)-d option** : It only prints the repeated lines - $uniq -d file.txt

iv) **-u option** : It prints only the unique lines - $uniq -u file.txt

e) job control

I)ps(process status)

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

II)kill

used to terminate processes manually - $ kill pid

OUTPUT

```
user@ubuntu:~$ sudo chown root:user text.txt
user@ubuntu:~$ ls -l
total 0
drwxr-xr-x 1 user user 4096 Aug  2 21:41 Docs
drwxr-xr-x 1 user user 4096 Aug  2 21:24 Downloads
drwxr-xr-x 1 user user 4096 Aug  2 21:24 Music
drwxr-xr-x 1 user user 4096 Aug  2 21:25 Pictures
-rw-r--r-- 1 root user    8 Aug  2 21:31 text.txt
user@ubuntu:~$ sudo adduser test
Adding user `test' ...
Adding new group `test' (1001) ...
Adding new user `test' (1001) with group `test' ...
Creating home directory `/home/test' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for test
Enter the new value, or press ENTER for the default
        Full Name []: test
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
user@ubuntu:~$ sudo chown root:test text.txt
user@ubuntu:~$ ls -l
total 0
drwxr-xr-x 1 user user 4096 Aug  2 21:41 Docs
drwxr-xr-x 1 user user 4096 Aug  2 21:24 Downloads
drwxr-xr-x 1 user user 4096 Aug  2 21:24 Music
drwxr-xr-x 1 user user 4096 Aug  2 21:25 Pictures
-rw-r--r-- 1 root test    8 Aug  2 21:31 text.txt
user@ubuntu:~$
```

```
user@ubuntu:~$ cat >exp1.txt
123
456
789
^Z
[4]+  Stopped                 cat > exp1.txt
user@ubuntu:~$ cat >exp2.txt
01001
10100
10100
^Z
[5]+  Stopped                 cat > exp2.txt
user@ubuntu:~$ cat exp1.txt exp2.txt
123
456
789
01001
10100
10100
```

```
user@ubuntu:~$ cat -n exp1.txt
     1  123
     2  456
     3  789
user@ubuntu:~$ cat >> exp1.txt
000
^Z
[6]+  Stopped                 cat >> exp1.txt
user@ubuntu:~$ cat exp1.txt
123
456
789
000
user@ubuntu:~$ cat < exp1.txt
123
456
789
000
```

```
user@ubuntu:~$ xyz 2> err.txt
user@ubuntu:~$ cat err.txt

Command 'xyz' not found, did you mean:

  command 'xz' from deb xz-utils (5.2.4-1ubuntu1)
  command 'xye' from deb xye (0.12.2+dfsg-9build1)

Try: sudo apt install <deb name>
```

```
user@ubuntu:~$ ls -l | more
total 0
drwxr-xr-x 1 user user 4096 Aug  2 21:41 Docs
drwxr-xr-x 1 user user 4096 Aug  2 21:24 Downloads
drwxr-xr-x 1 user user 4096 Aug  2 21:24 Music
drwxr-xr-x 1 user user 4096 Aug  2 21:25 Pictures
-rw-r--r-- 1 user user  177 Aug  2 23:33 err.txt
-rw-r--r-- 1 user user    6 Aug  2 22:49 exp.txt
-rw-r--r-- 1 user user   16 Aug  2 23:30 exp1.txt
-rw-r--r-- 1 user user   18 Aug  2 22:53 exp2.txt
-rw-r--r-- 1 root test    8 Aug  2 21:31 text.txt
user@ubuntu:~$ cat exp1.txt | sort
000
123
456
789
user@ubuntu:~$ head -n 2 exp1.txt
123
456
user@ubuntu:~$ head -c 8 exp1.txt
123
456
user@ubuntu:~$ tail -2 exp1.txt
789
000
```

```
user@ubuntu:~$ tail +2 exp1.txt
456
789
000
user@ubuntu:~$ tail -c -2 exp1.txt
0
user@ubuntu:~$ tail -c +4 exp1.txt

456
789
000
user@ubuntu:~$ head -q exp1.txt text.txt
123
456
789
000
abcdefg
user@ubuntu:~$ tail -q exp1.txt text.txt
123
456
789
000
abcdefg
user@ubuntu:~$ head -v exp1.txt
==> exp1.txt <==
123
456
789
000
user@ubuntu:~$ tail -v exp1.txt
==> exp1.txt <==
123
456
789
000
user@ubuntu:~$
```

```
user@ubuntu:~$ passwd
Changing password for user.
Current password:
New password:
Retype new password:
passwd: password updated successfully
user@ubuntu:~$
```

```
user@ubuntu:~$ wc exp1.txt
 4  4 16 exp1.txt
user@ubuntu:~$ wc exp1.txt exp2.txt
 4  4 16 exp1.txt
 3  3 18 exp2.txt
 7  7 34 total
user@ubuntu:~$ wc -l exp1.txt
4 exp1.txt
user@ubuntu:~$ wc -l exp1.txt exp2.txt
 4 exp1.txt
 3 exp2.txt
 7 total
user@ubuntu:~$ wc -w exp1.txt text.txt
 4 exp1.txt
 1 text.txt
 5 total
user@ubuntu:~$ wc -c exp1.txt text.txt
16 exp1.txt
 8 text.txt
24 total
user@ubuntu:~$ wc -m exp1.txt text.txt
16 exp1.txt
 8 text.txt
24 total
user@ubuntu:~$
```

```
user@ubuntu:~$ sort -n exp1.txt
000
123
456
789
user@ubuntu:~$ sort -nr exp1.txt
789
456
123
000
user@ubuntu:~$ sort -k2 exp1.txt
000
123
456
789
user@ubuntu:~$ sort -u exp5.txt
abc
asd
ghz
mnc
mno
user@ubuntu:~$
```

```
user@ubuntu:~$ uniq exp.txt
apple
banana
redapple
mango
pineapple
pineredapple
mango
orange
user@ubuntu:~$ uniq -c exp.txt
      2 apple
      1 banana
      1 redapple
      1 mango
      1 pineapple
      1 pineredapple
      1 mango
      1 orange
user@ubuntu:~$ uniq -d exp.txt
apple
user@ubuntu:~$ uniq -u exp.txt
banana
redapple
mango
pineapple
pineredapple
mango
orange
user@ubuntu:~$
```

```
user@ubuntu:~$ sed 's/apple/redapple/' exp.txt
redapple
redapple
banana
redredapple
mango
pineredapple
pineredredapple
mango
orange
user@ubuntu:~$ ps
  PID TTY          TIME CMD
  174 tty1     00:00:02 bash
  570 tty1     00:00:00 cat
  585 tty1     00:00:00 cat
  586 tty1     00:00:00 cat
  595 tty1     00:00:00 cat
  596 tty1     00:00:00 cat
  607 tty1     00:00:00 cat
  765 tty1     00:00:00 ps
user@ubuntu:~$ kill 174
```

RESULT

Familiarised with Linux Commands for Operation.

SHELL PROGRAMMING

Experiment No:4

Date : 07/06/2022

AIM :

a) Briefly explain the basics of shell scripting

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

b) Write shell script to show various system configuration like

1.Currently logged user and his login name
2. Your current shell
3. Your home directory
4. Your operating system type
5. Your current path setting
6. Your current working directory
7. Number of users currently logged in

Algorithm

1. currently logged user and his login name
   x=$(logname)
   echo "Currently logged user and his logname : $x"
2. Your current shell
   echo $SHELL
3. Your home directory
   echo $HOME
4. Your operating system type
   x=$(arch)
   echo "Your operating system type : $x"
5. Your current path setting

echo $PATH

6. Your current working directory

echo $pwd

7. Show Currently logged number of users

echo $users

SCRIPT

```
#!/bin/bash
x=$USER
shell=$SHELL
home=$HOME
type=$(arch)
path=$PATH
dir=$PWD
n=$(who -q)
echo "currently logged user and his logname :$x"
echo "current shell:$shell"
echo "home directory:$home"
echo "operating system type :$type"
echo "current path setting :$path"
echo "current working directory :$dir"
echo "number of users currently logged in :$n"
```

OUTPUT

RESULT

Program Finished and output obtained Successfully.

SHELL SCRIPT TO SHOW VARIOUS SYSTEM CONFIGURATION

Experiment No:5

Date : 07/06/2022

AIM :

To Write shell script to show various system configurations like

a) your OS and version, release number, kernel version
b) all available shells
c) computer CPU information like processor type, speed etc
d) memory information
e) hard disk information like size of hard-disk, cache memory, model etc
f) File system (Mounted)

ALGORITHM

a)your OS and version, release number, kernel version

```
x=$(lsb_release -a)
```

b) all available shells

```
cat /etc/shells
```

c) computer CPU information like processor type, speed etc

```
echo $(lscpu)
```

d) memory information

```
echo $(free -m)
```

e) hard disk information like size of hard-disk, cache memory, model etc

```
echo $(sudo dmidecode -t memory)
```

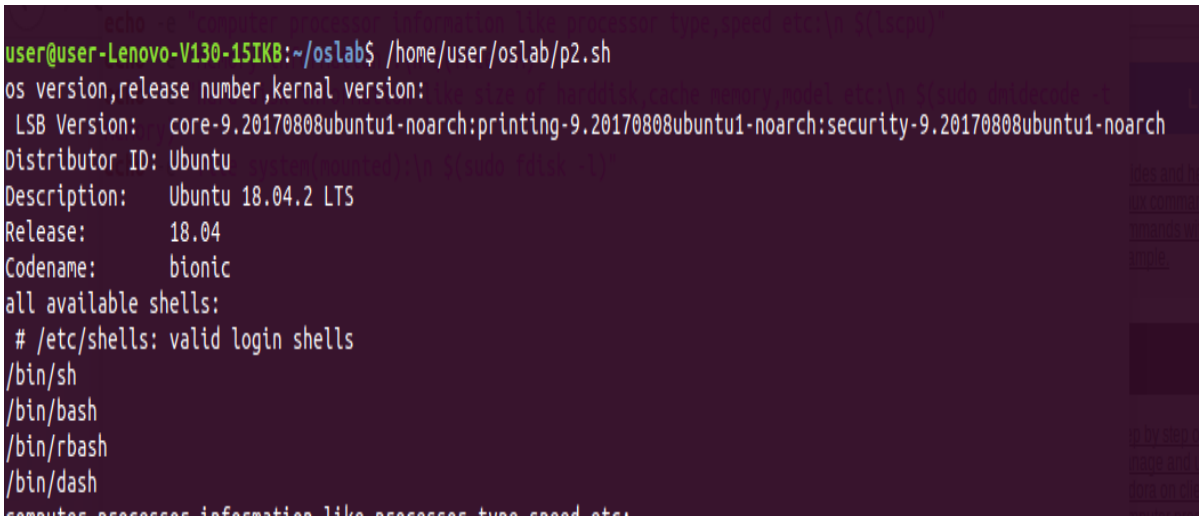
f) File system (Mounted)

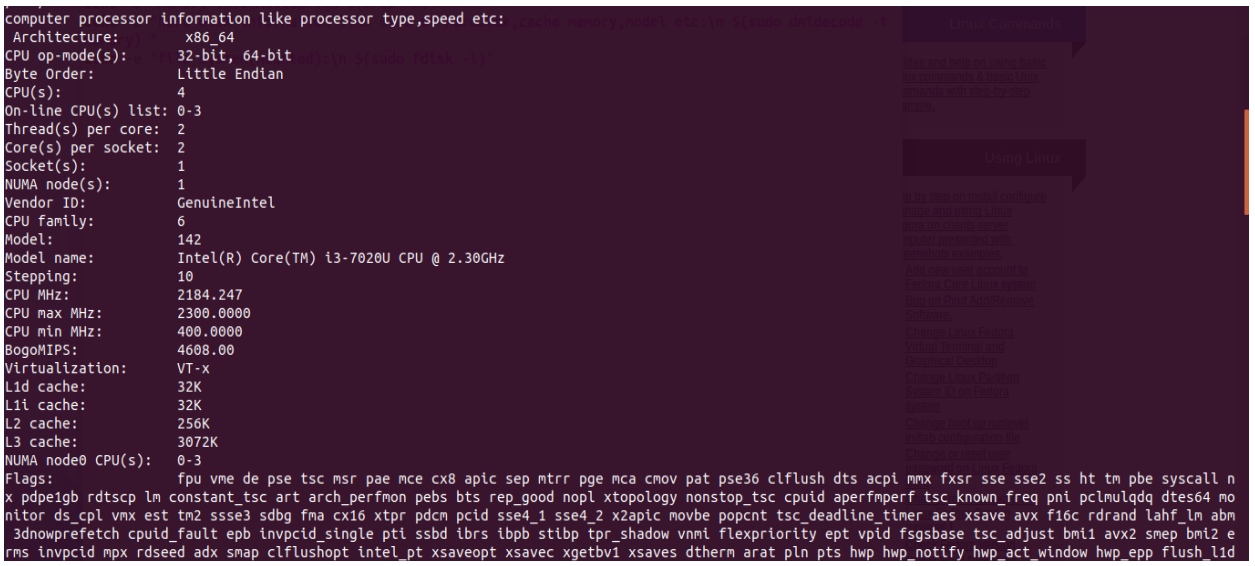
```
echo $(sudo fdisk -l)
```

SCRIPT

```
#!/bin/bash
echo -e "os version,release number,kernel version: \n $(lsb_release -a)"
echo -e "all available shells:\n $(cat /etc/shells)"
echo -e "computer processor information like processor type,speed etc:\n
$(lscpu)"
echo -e "memory information :\n $(free -m)"
echo -e "hard disk information like size of harddisk,cache memory,model etc:\n
$(sudo dmidecode -t memory) "
echo -e "file system(mounted):\n $(sudo fdisk -l)"
```

OUTPUT





```
memory information :
              total        used        free      shared  buff/cache   available
Mem:           3113        1292         476         130        1344        1464
Swap:         19071           0       19071
hard disk information like size of harddisk cache memory model etc:
```

```
hard disk information like size of harddisk,cache memory,model etc:
 # dmidecode 3.1
Getting SMBIOS data from sysfs.
SMBIOS 3.0.0 present.

Handle 0x0002, DMI type 16, 23 bytes
Physical Memory Array
        Location: System Board Or Motherboard
        Use: System Memory
        Error Correction Type: None
        Maximum Capacity: 32 GB
        Error Information Handle: Not Provided
        Number Of Devices: 1

Handle 0x0003, DMI type 17, 40 bytes
Memory Device
        Array Handle: 0x0002
        Error Information Handle: Not Provided
        Total Width: 64 bits
        Data Width: 64 bits
        Size: 4096 MB
        Form Factor: SODIMM
        Set: None
        Locator: ChannelA-DIMM0
        Bank Locator: BANK 0
        Type: DDR4
        Type Detail: Synchronous Unbuffered (Unregistered)
        Speed: 2400 MT/s
        Manufacturer: SK Hynix
        Serial Number: 00000000
        Asset Tag: 9876543210
        Part Number: HMA851S6AFR6N-UH
        Rank: 1
        Configured Clock Speed: 2400 MT/s
```

```
        Minimum Voltage: Unknown
        Maximum Voltage: Unknown
        Configured Voltage: 1.2 V

Handle 0x0004, DMI type 17, 40 bytes
Memory Device
        Array Handle: 0x0002
        Error Information Handle: Not Provided
        Total Width: Unknown
        Data Width: Unknown
        Size: No Module Installed
        Form Factor: Unknown
        Set: None
        Locator: ChannelB-DIMM0
        Bank Locator: BANK 2
        Type: Unknown
        Type Detail: None
        Speed: Unknown
        Manufacturer: Not Specified
        Serial Number: Not Specified
        Asset Tag: Not Specified
        Part Number: Not Specified
        Rank: Unknown
        Configured Clock Speed: Unknown
        Minimum Voltage: Unknown
        Maximum Voltage: Unknown
        Configured Voltage: Unknown
Partition 2 does not start on physical sector boundary.
```

```
file system(mounted):
 Disk /dev/loop0: 97.9 MiB, 102637568 bytes, 200464 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/loop1: 140.7 MiB, 147488768 bytes, 288064 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/loop2: 55.5 MiB, 58159104 bytes, 113592 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/loop3: 98.4 MiB, 103129088 bytes, 201424 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/loop4: 140.7 MiB, 147501056 bytes, 288088 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

```
Disk /dev/loop5: 64.8 MiB, 67915776 bytes, 132648 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/loop6: 55.4 MiB, 58073088 bytes, 113424 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/loop7: 62.1 MiB, 65105920 bytes, 127160 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes



Disk /dev/sda: 931.5 GiB, 1000204886016 bytes, 1953525168 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disklabel type: dos
Disk identifier: 0xd9fa2484

Device     Boot      Start        End    Sectors    Size Id Type
/dev/sda1  *          2048   97656344   97654297   46.6G 83 Linux
/dev/sda2         97658878  333006847  235347970  112.2G  5 Extended
/dev/sda5         97658880  136718335   39059456   18.6G 82 Linux swap / Solaris
/dev/sda6        137697280  333006847  195309568   93.1G 83 Linux
user@user-Lenovo-V130-15IKB:~/oslab$
```

RESULT

Program Finished and output obtained Successfully.

CALCULATOR

Experiment No:6

Date : 07/06/2022

AIM : To Write a shell script to implement a menu driven calculator with following functions

a) Addition
b) Subtraction
c) Multiplication
d) Division
e) Modulus

ALGORITHM

1. read two numbers a and b
2. initialise c=1, result=0
3. while [$c=1]
    do
        echo "1.Addition"
        echo "2.Subtraction"
        echo "3.Multiplication"
        echo "4.Division"
        echo "5.Modulus"
        echo "Enter your choice:"
4. read ch
    if ch=1, perform addition
    if ch=2, perform subtraction
    if ch=3, perform multiplication
    if ch=4, perform division
    if ch=5, perform modulus
   else print invalid choice
end
5.if c=0, exit

SCRIPT

```
#!/bin/bash
echo "enter first number : "
read a
echo -e "enter second number : "
read b
c=1
result=0
while [ $c=1 ]
do
echo "1.Addition"
echo "2.Subtraction"
echo "3.Multiplication"
echo "4.Division"
echo "5.Modulus"
echo "Enter your choice:"
read ch
case $ch in
1) result=$(($a+$b))
echo "The sum is :"
echo $result;;
2) result=$(($a-$b))
echo "The difference is :"
echo $result;;
3)result=$(($a*$b))
echo "The product is :"
echo $result;;
4)result=$(($a/$b))
echo "The quotient is :"
echo $result;;
5)result=$(($a%$b))
echo "Modulus is :"
echo $result;;
*) echo "enter a valid choice";;
esac
echo "Do you want to continue?(1 for yes and 0 for no)"
read c
if [ $c != 1 ]
```

```
then
        exit
fi
done
```

OUTPUT

```
user@LAPTOP-BPURTOLO:~$ ./calc.sh
enter first number :
25
enter second number :
5
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Modulus
Enter your choice:
3
The product is :
125
Do you want to continue?(1 for yes and 0 for no)
1
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Modulus
Enter your choice:
5
Modulus is :
0
Do you want to continue?(1 for yes and 0 for no)
```

```
user@LAPTOP-BPURTOLO:~$ ./calc.sh
enter first number :
24
enter second number :
5
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Modulus
Enter your choice:
5
Modulus is :
4
Do you want to continue?(1 for yes and 0 for no)
```

RESULT

Program finished and output obtained successfully.

Implement various text processing using grep

Experiment No:7

Date : 07/06/2022

AIM : To Implement various text processing using grep

DESCRIPTION

grep stands for global regular expression print. It is a command line utility for searching plain-text data sets for lines that match a regular expression.It search for PATTERN in each FILE
grep ‘STRING’ filename
The above command searches for a file for STRING and list the lines that contain a match
i) **$grep -i "string" filename.txt** : to search for a string case insensitively in the give file
ii)**$grep -c "string" filename.txt** : displaying the count of number of matches
iii**)$ grep -w "word" filename**: checking for the whole words in a file
iv)**$ grep -o "word" filename** : displaying only the matched pattern
v**)$ grep -n "word" filename** : Show line number while displaying the output
vi**)$ grep "^string" filename** : Matching the lines that start with a string
vii**)$ grep "string$" filename** : Matching the lines that end with a string
viii**)$grep -A numberoflines(n) “word” filename** : prints the searched line and n lines after the result
ix**)$grep -B numberoflines(n) “word” filename**: prints the searched line and n lines before the result
x**)$grep -C numberoflines(n) “word” filename** : prints the searched line and n lines after and before the result

OUTPUT

```
user@user-Lenovo-V130-15IKB:~/oslab$ cat exp1.txt
mango is a fruit
tomato is a vegetable
apple is red
mango is yellow
apple is a fruit
tomato is red
carrot is orange
orange is a fruit
CARROT is a vegetable
user@user-Lenovo-V130-15IKB:~/oslab$ grep "mango" exp1.txt
mango is a fruit
mango is yellow
user@user-Lenovo-V130-15IKB:~/oslab$ grep -i "CaRRoT" exp1.txt
carrot is orange
CARROT is a vegetable
user@user-Lenovo-V130-15IKB:~/oslab$ grep -c "fruit" exp1.txt
3
user@user-Lenovo-V130-15IKB:~/oslab$ grep -w "apple" exp1.txt
apple is red
apple is a fruit
user@user-Lenovo-V130-15IKB:~/oslab$ grep -o "CARROT" exp1.txt
CARROT
user@user-Lenovo-V130-15IKB:~/oslab$ grep -n "fruit" exp1.txt
1:mango is a fruit
5:apple is a fruit
8:orange is a fruit
user@user-Lenovo-V130-15IKB:~/oslab$ grep "^or" exp1.txt
orange is a fruit
user@user-Lenovo-V130-15IKB:~/oslab$ grep "uit$" exp1.txt
mango is a fruit
apple is a fruit
orange is a fruit
```

```
user@user-Lenovo-V130-15IKB:~/oslab$ grep -A2 "red" exp1.txt
apple is red
mango is yellow
apple is a fruit
tomato is red
carrot is orange
orange is a fruit
user@user-Lenovo-V130-15IKB:~/oslab$ grep -B1 "apple" exp1.txt
tomato is a vegetable
apple is red
mango is yellow
apple is a fruit
user@user-Lenovo-V130-15IKB:~/oslab$ grep -C1 "carrot" exp1.txt
tomato is red
carrot is orange
orange is a fruit
user@user-Lenovo-V130-15IKB:~/oslab$
```

RESULT

Program finished and output obtained successfully.

IMPLEMENT opendir, readdir, closedir IN LINUX OPERATING SYSTEM USING C

Experiment No : 8

Date : 20/06/2022

AIM: To illustrate the system calls of the Linux operating system: opendir, readdir, closedir)

ALGORITHM

1. Start the program
2. Create struct direct
3. declare the variable a and pointer dptr
4. Get the directory name
5. Open the directory
6. Read the contents in the directory and print it
7. Close the directory

PROGRAM

```c
#include <stdio.h>
#include <dirent.h>

int main(){
   struct dirent *de;
   DIR *dr = opendir(".");

   if(dr == NULL){
       printf("Directory not found.");
       return 0;
   }

   while((de = readdir(dr)) != NULL){
       printf("%s\n",de->d_name);
   }
```

```
    closedir(dr);
    return 0;
}
```

OUTPUT

```
user@user-HP-Laptop-15q-ds0xxx:/media/user/288C19CB8C199482/Misc/C/oslab$ gcc oslab1.c
user@user-HP-Laptop-15q-ds0xxx:/media/user/288C19CB8C199482/Misc/C/oslab$ ./a.out
.
..
a.out
oslab1.c
user@user-HP-Laptop-15q-ds0xxx:/media/user/288C19CB8C199482/Misc/C/oslab$
```

Result:

Program finished and output obtained successfully.

IMPLEMENT fork, getpid, exit IN LINUX OPERATING SYSTEM USING C

Experiment No: 9

Date : 20/06/2022

AIM: To write a c program to illustrate the following system calls of the linux operating system(fork, getpid, exit)

ALGORITHM

1. Start the program
2. Declare the variables pid,pid1,pid2
3. Call fork() system call to create process
4. If pid<0, exit
5. If pid=0 , get the child process id using getpid()
6. If pid>0, get the parent process id using getpid()
7. Print the process id
8. Stop the program

PROGRAM

```c
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
void main(){
   int pid,pidfork;
   pid = getpid();
   printf("ID of parent process: %d\n",pid);
   fork();
   pidfork = getpid();
   printf("ID of forked process: %d\n",pidfork);
   exit(0);
}
```

OUTPUT

```
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ gcc fork.c
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ ./a.out
ID of parent process: 10106
ID of forked process: 10106
ID of forked process: 10107
user@user-HP-Laptop-15q-ds0xxx:~/oslab$
```

Result:

Program finished and output obtained successfully.

SIMULATE CPU SCHEDULING ALGORITHMS(FCFS, SJF, RR, PRIORITY) AND FIND TURNAROUND TIME AND WAITING TIME USING C

Experiment No : 10

Date : 28/06/2022

AIM: To write a c program to simulate the following CPU scheduling algorithms to find turnaround time and waiting time.

a)FCFS b)SJF c)Round Robin d)Priority

a) FCFS

ALGORITHM

1. Declare the variables
2. Declare the variable i,n as integer,totalwtime and total time is equal to zero
3. Get the value of btime[i]
4. Assign wtime[0] as zero and tatime[0] as btime[0] and inside the loop calculate waiting time and turnaround time.
5. Calculate total waiting time and total turnaround time and calculate average waiting time and turnaround time by dividing it by the total number of process
6. Print total waiting time, total turnaround time,average waiting time and average turnaround time
7. Stop the program

PROGRAM

```
#include<stdio.h>

void waitingtime(int btime[],int wtime[],int n){
   int i;
   wtime[0] = 0;
   for(i=1;i<n;i++)
       wtime[i] = wtime[i-1] + btime[i-1];
}

void tatime(int tat[],int btime[],int wtime[],int n){
   int i;
   for(i=0;i<n;i++)
       tat[i] = btime[i] + wtime[i];
}

void avgtime(int proc[],int btime[],int n){
   int wtime[n],tat[n],i,avgwtime = 0,avgtatime = 0;
   waitingtime(btime,wtime,n);
   tatime(tat,btime,wtime,n);
   printf("Process Burst Waiting Turn Around\n");
   for(i=0;i<n;i++){
       avgwtime += wtime[i];
       avgtatime += tat[i];
       printf("%d \t %d \t %d \t
%d\n",proc[i],btime[i],wtime[i],tat[i]);
   }
   printf("Average waiting time: %f\n",(float)avgwtime/(float)n);
   printf("Average turn around time: %f\n",(float)avgtatime/(float)n);
}

void main(){
   int proc[] = {1,2,3};
   int btime[] = {5,8,12};
   int n = sizeof proc/sizeof proc[0];
   avgtime(proc,btime,n);
}
```

OUTPUT

```
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ gcc fcfs.c
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ ./a.out
Process Burst Waiting Turn Around
1        5       0       5
2        8       5       13
3        12      13      25
Average waiting time: 6.000000
Average turn around time: 14.333333
user@user-HP-Laptop-15q-ds0xxx:~/oslab$
```

b) Round Robin

ALGORITHM

1. Declare the variables
2. Declare the variable i,j as integer, totalwtime and totalttime is equal to zero
3. Get the number of processes n and time quantum
4. Inside the for loop get the value of burst time and arrival time
5. Check burst time of process is greater than time quantum or not
6. Calculate waiting time and turnaround time of processes
7. Calculate the total of waiting time and turnaround time and find average of waiting time and turnaround time by dividing it by number of processes
8. Stop the program

PROGRAM

```
#include<stdio.h>

typedef struct
{
   int no,at,bt,tat,wt;
}process;

process p[10];
int n,q[15];
int front =-1, rear =-1;

void insert(int p)
{
   if(front ==-1)
      front = 0;
   rear = rear + 1;
   q[rear] = p;
}

int delete(void)
{
   int p;
   p = q[front];
   front = front + 1;
   return p;
}

int main()
{
   int i, temp[10], exist[10]={0},a,t;
   int total_tat=0, total_wt=0, time =0;
   float avg_tat, avg_wt;
   printf("Enter the no: of processes : ");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
       printf("Enter arrival time burst time of process P%d : ",i);
      scanf("%d%d",&p[i].at,&p[i].bt);
      p[i].no = i;
      temp[i] = p[i].bt;
```

```
   }
   printf("Enter the time quantum : ");
   scanf("%d",&t);
   insert(0);
   exist[0] = 1;
   while(front<=rear)
   {
      a = delete();
      if(p[a].bt>=t)
      {
         p[a].bt = p[a].bt-t;
         time = time+t;
      }
      else
      {
         time = time+p[a].bt;
         p[a].bt = 0;
      }
      for(i=0;i<n;i++)
      {
         if(exist[i]==0 && p[i].at<=time)
         {
            insert(i);
            exist[i] = 1;
         }
      }
      if(p[a].bt==0)
      {
         p[a].tat = time - p[a].at;
         p[a].wt = p[a].tat - temp[a];
         total_tat = total_tat + p[a].tat;
         total_wt = total_wt+p[a].wt;
       }
      else
         insert(a);
   }
   avg_tat = (float)total_tat/n;
   avg_wt = (float)total_wt/n;
   printf("Process\tArrival time\tBurst time\tTurn around time\tWaiting
time\n");
   for(i=0;i<n;i++)
   {
```

```
printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i,p[i].at,temp[i],p[i].tat,p[i].
wt);
    }
    printf("Average turnaround time = %.2f\n",avg_tat);
    printf("Average waiting time = %.2f\n",avg_wt);
    return 0;
}
```

OUTPUT

```
Enter the no: of processes : 3
Enter arrival time burst time of process P0 : 0 3
Enter arrival time burst time of process P1 : 1 4
Enter arrival time burst time of process P2 : 2 5
Enter the time quantum : 2
Process Arrival time    Burst time      Turn around time        Waiting time
P0              0               3               7               4
P1              1               4               8               4
P2              2               5               10              5
Average turn around time = 8.33
Average waiting time = 4.33
```

c) SJF

ALGORITHM

1. Declare the variables
2. Declare the variable i,j as integer,totaltatime and totalwtime is equal to zero
3. Get the value of n and assign burst time for each process
4. Assign wtime[0] as zero and tatime[0] as btime[0] and inside the loop calculate wait time and turnaround time
5. Calculate total waiting time and total turnaround time and calculate average waiting time and average turnaroundtime by dividing it by total number of process
6. Print total waiting time, total turnaround time, average waiting time, and average turnaround time
7. Stop the program

PROGRAM

```
#include<stdio.h>
struct process
{
    int WT,AT,BT,TAT;
};

struct process a[10];

int main()
{
    int n,temp[10];
    int count=0,t=0,short_P;
    float total_WT=0, total_TAT=0,Avg_WT,Avg_TAT;
    printf("Enter the number of the process\n");
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
printf("Enter the arrival time and burst time of the P%d : ",i);
        scanf("%d %d",&a[i].AT,&a[i].BT);

        // copying the burst time in
        // a temp array for the further use
        // in calculation of WT
        temp[i]=a[i].BT;
    }

    // we initialise the burst time
    // of a process with the maximum
    a[9].BT=10000;
    // loop will be execute until all the process
    // complete so we use count!= number of
    // the process
    for(t=0;count!=n;t++)
    {
        // for finding min burst
        // it is useful
        short_P=9;
        for(int i=0;i<n;i++)
        {
            if(a[i].BT<a[short_P].BT && (a[i].AT<=t && a[i].BT>0))
```

```
            {
                short_P=i;
            }

        }

        a[short_P].BT=a[short_P].BT-1;

        // if any process is completed
        if(a[short_P].BT==0)
        {
            // one process complete
            count++;
            a[short_P].WT=t+1-a[short_P].AT-temp[short_P];
            a[short_P].TAT=t+1-a[short_P].AT;

            // total calculation
            total_WT=total_WT+a[short_P].WT;
            total_TAT=total_TAT+a[short_P].TAT;
        }



    }
    Avg_WT=total_WT/n;
    Avg_TAT=total_TAT/n;

    // printing of the answer
    printf("Process | Burst Time | Arrival Time | Waiting Time | Turn
Around Time\n");
    for(int i=0;i<n;i++)
    {

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i,a[i].BT,a[i].AT,a[i].WT,a[i].TA
T);
    }
    printf("Avg waiting time of the process is %f\n",Avg_WT);
    printf("Avg turn around time of the process %f\n",Avg_TAT);

}
```

OUTPUT

```
Enter the number of the process
3
Enter the arrival time and burst time of the P0 : 14 56
Enter the arrival time and burst time of the P1 : 16 28
Enter the arrival time and burst time of the P2 : 85 20
Process | Burst Time | Arrival Time | Waiting Time | Turn Around Time
0               56              14              28              84
1               28              16              0               28
2               20              85              13              33
Avg waiting time of the process is 13.666667
Avg turn around time of the process 48.333332
```

d) Priority

ALGORITHM

1. Declare the variables
2. Declare the variable i,j as integer, totalwtime and total time is equal to zero
3. Get the value of n as number of processes
4. Inside the for loop get the value of burst time and priority for each process
5. Assign wtime[0] as zero and tatime[0] as btime[0]
6. Check priority[i] is greater than priority[j] and sort
7. Calculate the total turnaround time and waiting time and find the average waiting time and average turnaround time by dividing it by number of processes
8. Stop the program

PROGRAM

```
Priority
#include<stdio.h>

typedef struct
{
   int no,at,bt,tat,wt,pt;
}process;

process p[10];

int main()
{
   int i,n,temp[10],t,count=0,sp;
   int total_tat=0, total_wt=0;
   float avg_tat, avg_wt;
   printf("Enter the no: of processes : ");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("Enter arrival time, burst time and priority of process
P%d : ",i);
      scanf("%d%d%d",&p[i].at,&p[i].bt,&p[i].pt);
      p[i].no = i;
      temp[i] = p[i].bt;
   }
   p[9].pt = 1000000;
   for(t=0;count!=n;t++)
   {
      sp = 9;
      for(i=0;i<n;i++)
      {
         if(p[sp].pt>p[i].pt && p[i].at<=t && p[i].bt>0)
         {
            sp =i;
         }
      }
      p[sp].bt = p[sp].bt-1;
      if(p[sp].bt==0)
```

```
      {
         count++;
         p[sp].tat = t+1-p[sp].at;
         p[sp].wt = p[sp].tat - temp[sp];
         total_tat +=p[sp].tat;
         total_wt+=p[sp].wt;
      }
   }
   avg_tat = (float)total_tat/n;
   avg_wt = (float)total_wt/n;
   printf("Process\tArrival time\tBurst time\tPriority\tTurn around
time\tWaiting time\n");
   for(i=0;i<n;i++)
   {

printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i,p[i].at,temp[i],p[i].pt,
p[i].tat,p[i].wt);
   }
   printf("Average turn around time = %.2f\n",avg_tat);
   printf("Average waiting time = %.2f\n",avg_wt);
   return 0;
}
```

OUTPUT

```
Enter the no: of processes : 3
Enter arrival time, burst time and priority of process P0 : 0 3 2
Enter arrival time, burst time and priority of process P1 : 1 4 3
Enter arrival time, burst time and priority of process P2 : 2 3 1
Process Arrival time    Burst time      Priority        Turn around time        Waiting time
P0              0               3               2               6               3
P1              1               4               3               9               5
P2              2               3               1               3               0
Average turn around time = 6.00
Average waiting time = 2.67
```

Result:

Program finished and output obtained successfully.

SIMULATE BANKERS ALGORITHM FOR DEADLOCK AVOIDANCE USING C

Experiment No : 11

Date : 02/08/2022

AIM: To implement banker's algorithm for deadlock avoidance

ALGORITHM

1. Start the program
2. Declare the variables
3. Read the number of process, resources, available resources, allocation matrix and max matrix
4. Find need matrix
5. Compare each and every process using the banker's algorithm
6. If the process is in safe state then it is a not a deadlock process otherwise it is a deadlock process
7. produce the safe sequence
8. Stop the program

PROGRAM

```
#include<stdio.h>

void main(){
   int
n,m,i,j,k,alloc[10][10],max[10][10],need[10][10],flag[10],available[10]
,fla = 0;
   printf("Enter the number of processes: ");
   scanf("%d",&n);
```

```
printf("Enter the number of resources: ");
scanf("%d",&m);
printf("Enter the current allocations:\n");
for(i=0;i<n;i++){
    printf("Enter current allocations for P%d: ",i);
    for(j=0;j<m;j++){
        scanf("%d",&alloc[i][j]);
    }
}
printf("Enter the maximum allocations:\n");
for(i=0;i<n;i++){
    printf("Enter maximum allocations for P%d: ",i);
    for(j=0;j<m;j++){
        scanf("%d",&max[i][j]);
    }
}
for(i=0;i<n;i++){
    flag[i] = 0;
    for(j=0;j<m;j++){
        need[i][j] = max[i][j] - alloc[i][j];
    }
}
printf("Enter the available instances: ");
for(i=0;i<m;i++){
    scanf("%d",&available[i]);
}
printf("The process sequence is:\n");
for(i=0;i<n;i++){
    for(j=0;j<n;j++){
        if(flag[i] == 0){
            for(k=0;k<m;k++){
                if(need[j][k]>available[k]){
                    fla = 1;
                    break;
                }
            }
            if(fla == 0){
                for(k=0;k<m;k++){
                    available[k]+= alloc[j][k];
                }
                flag[i] = 1;
                printf("P%d ",i);
            }
```

```
            }
        }
    }
}
```

OUTPUT

```
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ gcc bankers.c
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ ./a.out
Enter the number of processes: 5
Enter the number of resources: 4
Enter the current allocations:
Enter current allocations for P0: 0 0 1 2
Enter current allocations for P1: 1 0 0 0
Enter current allocations for P2: 1 3 5 4
Enter current allocations for P3: 0 6 3 2
Enter current allocations for P4: 0 0 1 4
Enter the maximum allocations:
Enter maximum allocations for P0: 0 0 1 2
Enter maximum allocations for P1: 1 7 5 0
Enter maximum allocations for P2: 2 3 5 6
Enter maximum allocations for P3: 0 6 5 2
Enter maximum allocations for P4: 0 6 5 6
Enter the available instances: 1 5 2 0
The process sequence is:
P0 P1 P2 P3 P4 user@user-HP-Laptop-15q-ds0xxx:~/oslab$ ./a.out
```

Result:

Program finished and output obtained successfully.

ILLUSTRATE CREATION OF SHARED MEMORY USING C

Experiment No: 12

Date: 02/08/2022

AIM: To illustrate the creation of shared memory

ALGORITHM

1. Include all required header files
2. key = ftok(".",'s')
3. shmid =shmget(key,SEGSIZE,IPC_CREAT|IPC_EXCL|0666)
4. if(shmid = -1)
    a. perror("shmget")

b. exit()

5. Else

a. Print(shmid)

6. End if
7. segptr = (char*)shmat(shmid,0,0)
8. if(segptr = (char*) - 1)
    a. perror("shmat")
    b. exit()
9. End if
10. Read buff
11. strcpy(segptr,buff)
12. Print (segptr)

13. if(shmctl(shmid,IPC_RMID,0) = -1)
    a. Print(Can't remove shared memory segment..)
14. Else
    a. Print(Removed successfully.)
15. End if

PROGRAM

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

int main(){
   void *sharedmem;
   char buff[100];
   int shmid;
   shmid = shmget((key_t)1122,1024,0666|IPC_CREAT);
   printf("Key of shared memory: %d\n",shmid);
   sharedmem = shmat(shmid,NULL,0);
   printf("Process attached at: %p\n Enter data to write in shared
memory: \n",sharedmem);
   read(0,buff,100);
   strcpy(sharedmem,buff);
   printf("You wrote: %s\n",(char *)sharedmem);
   shmid = shmget((key_t)1122,1024,0666);
   printf("Key of shared memory: %d\n",shmid);
   sharedmem = shmat(shmid,NULL,0);
   printf("Process attached at: %p\n",sharedmem);
   printf("Data written in shared memory: %s\n",(char *)sharedmem);
}
```

OUTPUT

```
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ gcc ipc_sender.c
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ ./a.out
Key of shared memory: 2752526
Process attached at: 0x7f4f7f754000
 Enter data to write in shared memory:
hello world!
You wrote: hello world!

Key of shared memory: 2752526
Process attached at: 0x7f4f7f753000
Data written in shared memory: hello world!
```

Result:

Program finished and output obtained successfully.

SIMULATE PRODUCER–CONSUMER PROBLEM USING SEMAPHORES USING C

Experiment : 13

Date : 02/08/2022

AIM: To simulate producer-consumer problems using semaphores

ALGORITHM

Producer

do{

//produce an item

wait(empty);

wait(mutex);

//place in buffer

signal(mutex);

signal(full);

}while(true)

Consumer

do{

wait(full);

wait(mutex);

// remove item from buffer

signal(mutex);

signal(empty);

// consumes item

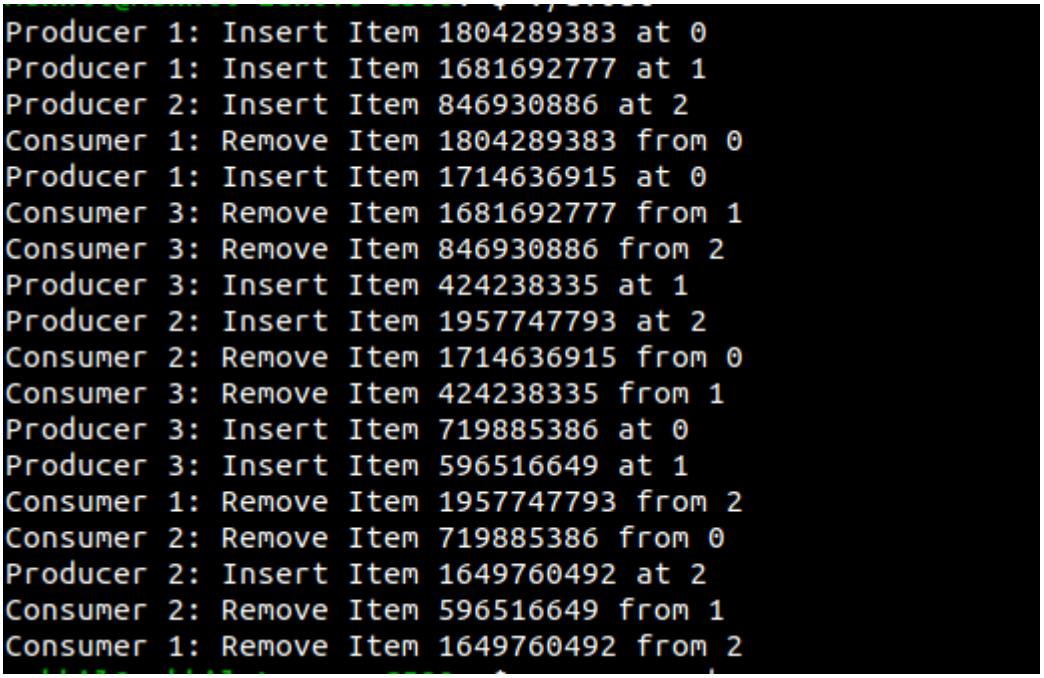
}while(true)

PROGRAM

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


#define MaxItems 5
#define BufferSize 5 sem_t empty;

sem_t full;
int in = 0;
int out = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;

void *producer(void *pno)
{
    int item;
    for(int i = 0; i < MaxItems; i++) {
        item = rand(); // Produce an random item
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[in] = item;
        printf("Producer %d: Insert Item %d at %d\n", *((int
*)pno),buffer[in],in);
        in = (in+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
    }
}
void *consumer(void *cno)
{
    for(int i = 0; i < MaxItems; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int item = buffer[out];
```

```
        printf("Consumer %d: Remove Item %d from %d\n",*((int
*)cno),item, out);
        out = (out+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
    }
}

int main()
{

    pthread_t pro[5],con[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty,0,BufferSize);
    sem_init(&full,0,0);

    int a[5] = {1,2,3,4,5}; //Just used for numbering the producer and
consumer

    for(int i = 0; i < 5; i++) {
        pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
    }
    for(int i = 0; i < 5; i++) {
        pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
    }

    for(int i = 0; i < 5; i++) {
        pthread_join(pro[i], NULL);
    }
    for(int i = 0; i < 5; i++) {
        pthread_join(con[i], NULL);
    }

    pthread_mutex_destroy(&mutex);
    sem_destroy(&empty);
    sem_destroy(&full);

    return 0;

}
```

OUTPUT



Result:

Program finished and output obtained successfully.

SIMULATE DISK SCHEDULING ALGORITHMS(FCFS, SCAN, C-SCAN) USING C

EXPERIMENT NO : 14

DATE : 02/08/2022

AIM: To simulate the following disk scheduling algorithms.

a) FCFS b) SCAN c) C-SCAN

a) FCFS

ALGORITHM

1. Let us one by one take the tracks in default order and calculate the absolute distance of the track from the head.
2. Increment the total seek count with this distance.
3. Currently serviced track position now becomes the new head position.
4. Go to step 2 until all tracks in the request array have not been serviced.
5. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. ‘head’ is the position of disk head.

PROGRAM

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

int req[10];

void fcfs(int req[],int n,int head){
   int curr,i,distance,seek = 0;
   for(i=0;i<n;i++){
       curr = req[i];
       distance = abs(head-curr);
       seek += distance;
       head = curr;
   }
```

```
    printf("Total seek count is: %d\n",seek);
}

void main(){
    int n,i,head;
    printf("Enter the number of requests: ");
    scanf("%d",&n);
    printf("Enter the request sequence: ");
    for(i=0;i<n;i++){
        scanf("%d",&req[i]);
    }
    printf("Enter head: ");
    scanf("%d",&head);
    fcfs(req,n,head);
}
```

OUTPUT

```
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ ./a.out
Enter the number of requests: 8
Enter the request sequence: 176 79 34 60 92 11 41 114
Enter head: 50
Total seek count is: 510
user@user-HP-Laptop-15q-ds0xxx:~/oslab$
```

b) SCAN

ALGORITHM

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. ‘head’ is the position of disk head.
2. Let direction represents whether the head is moving towards left or right.
3. In the direction in which the head is moving, service all tracks one by one.

4. Calculate the absolute distance of the track from the head.
5. Increment the total seek count with this distance.
6. Currently serviced track position now becomes the new head position.
7. Go to step 3 until we reach one of the ends of the disk.
8. If we reach the end of the disk, reverse the direction and go to step 2 until all tracks in the request array have not been serviced.

PROGRAM

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

void main(){
   int
queue[25],n,headposition,i,j,k,seek=0,maxrange,difference,temp,queue1[2
0],queue2[20],temp1=0,temp2=0;
   printf("Enter the maximum range of Disk: ");
   scanf("%d",&maxrange);
   printf("Enter the number of queue requests: ");
   scanf("%d",&n);
   printf("Enter the initial head position: ");
   scanf("%d",&headposition);
  printf("Enter the disk positions to be read(queue): ");
   for(i=1;i<=n;i++){
       scanf("%d",&temp);
       if(temp>headposition){
           queue1[temp1]=temp;
           temp1++;
       }
       else{
           queue2[temp2]=temp;
           temp2++;
       }
   }
   for(i=0;i<temp1-1;i++){
       for(j=i+1;j<temp1;j++){
           if(queue1[i]>queue1[j]){
               temp=queue1[i];
               queue1[i]=queue1[j];
```

```
                queue1[j]=temp;
            }
        }
    }
    for(i=0;i<temp2-1;i++){
        for(j=i+1;j<temp2;j++){
            if(queue2[i]<queue2[j]){
                temp=queue2[i];
                queue2[i]=queue2[j];
                queue2[j]=temp;
            }
        }
    }
    for(i=1,j=0;j<temp1;i++,j++){
        queue[i]=queue1[j];
    }
    queue[i]=maxrange;
    for(i=temp1+2,j=0;j<temp2;i++,j++){
        queue[i]=queue2[j];
    }
    queue[i]=0;
    queue[0]=headposition;
    for(j=0; j<=n; j++) {
        difference = abs(queue[j+1]-queue[j]);
        seek = seek + difference;
    }
    printf("Total Seek Time= %d\n", seek);

}
```

OUTPUT

```
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ gcc scan.c
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ ./a.out
Enter the maximum range of Disk: 199
Enter the number of queue requests: 7
Enter the initial head position: 50
Enter the disk positions to be read(queue): 82 170 43 140 24 16 190
Total Seek Time= 332
```

c) C-SCAN

ALGORITHM

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. ‘head’ is the position of disk head.
2. The head services only in the right direction from 0 to size of the disk.
3. While moving in the left direction do not service any of the tracks.
4. When we reach the beginning(left end) , reverse the direction.
5. While moving in the right direction it services all tracks one by one.
6. While moving in the right direction, calculate the absolute distance of the track from the head.
7. Increment the total seek count with this distance.
8. Currently serviced track position now becomes the new head position.
9. Go to step 6 until we reach the right end of the disk.
10. If we reach at the right end of the disk, reverse the direction and go to step 3 until all tracks in the request array have not been serviced.

PROGRAM

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

void main(){
   int size,n,i,j,temp,head,left[10],right[10],temp1 = 1,temp2 =
1,distance,seek = 0;
   printf("Enter the disk size: ");
   scanf("%d",&size);
```

```
printf("Enter the number of requests: ");
scanf("%d",&n);
printf("Enter the request sequence: ");
left[0] = 0;
right[0] = size;
for(i=0;i<n;i++){
    scanf("%d",&temp);
    if(temp<head){
        left[temp1] = temp;
        temp1++;
    }
    else{
        right[temp2] = temp;
        temp2++;
    }
}
printf("Enter head position: ");
scanf("%d",&head);
//sort left array
for(i=0;i<temp1-1;i++){
    for(j=0;j<temp1-i-1;j++){
        if(left[j]>left[j+1]){
            temp = left[j];
            left[j] = left[j+1];
            left[j+1] = temp;
        }
    }
}
//sort right array
for(i=0;i<temp2-1;i++){
    for(j=0;j<temp2-i-1;j++){
        if(right[j]>right[j+1]){
            temp = right[j];
            right[j] = right[j+1];
            right[j+1] = temp;
        }
    }
}
for(i=0;i<temp2;i++){
    distance = abs(head - right[i]);
    seek += distance;
    head = right[i];
}
```

```
    head = 0;
    seek += size;
    for(i=0;i<temp1;i++){
        distance = abs(head - left[i]);
        seek += distance;
        head = left[i];
    }
    printf("Total seek: %d\n",seek);
}
```

OUTPUT

```
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ gcc cscan.c
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ ./a.out
Enter the disk size: 199
Enter the number of requests: 8
Enter the request sequence: 176 79 34 60 92 11 41 114
Enter head position: 50
Total seek: 426
```

Result:

Program finished and output obtained successfully.

IMPLEMENT PAGE REPLACEMENT ALGORITHMS(FIFO, LRU, LFU) USING C

Experiment No: 15

Date : 02/08/2022

AIM: To implement the following page replacement algorithms.

a) FIFO b) LRU c) LFU

a) FIFO

ALGORITHM

1- Start traversing the pages.

i) If the set holds less pages than capacity.

a) Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.

b) Simultaneously maintain the pages in the queue to perform FIFO.

c) Increment page fault

ii) Else

If the current page is present in set, do nothing.

Else

a) Remove the first page from the queue as it was the first to be entered in the memory

b) Replace the first page in the queue with

the current page in the string.

c) Store the current page in the queue.

d) Increment page faults.

2. Return page faults.

PROGRAM

```
#include<stdio.h>

void main(){
   int n,i,j,req[10],cap,set[5],c = 0,pfault = 0,q[10],qb = 0,qf =
0,flag = 0,val;
   printf("Enter the number of requests: ");
   scanf("%d",&n);
   printf("Enter the page sequence: ");
   for(i=0;i<n;i++){
       scanf("%d",&req[i]);
   }
   printf("Enter the capacity of the memory: ");
   scanf("%d",&cap);
   for(i=0;i<n;i++){
       if(c<cap){
          set[c] = req[i];
          c++;
          pfault++;
          q[qb] = req[i];
          qb++;
       }
       else{
           for(j=0;j<cap;j++){
               if(set[j] == req[i]){
                   flag = 1;
                   break;
               }
           }



           if(flag == 0){
               val = q[qf];
               qf++;
```

```
                for(j=0;j<cap;j++){
                    if(set[j] == val){
                        set[j] = req[i];
                        pfault++;
                        break;
                    }}}}}
   printf("The number of page faults is: %d\n",pfault);
}
```

OUTPUT

```
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ gcc fifo.c
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ ./a.out
Enter the number of requests: 5
Enter the page sequence: 7 0 1 2 0
Enter the capacity of the memory: 3
The number of page faults is: 4
```

b) LRU

ALGORITHM

1- Start traversing the pages.

i) If the set holds less pages than capacity.

a) Insert page into the set one by one until

the size of set reaches capacity or all

page requests are processed.

b) Simultaneously maintain the recent occurred

index of each page in a map called indexes.

c) Increment page fault

ii) Else

If the current page is present in set, do nothing.

Else

a) Find the page in the set that was least

recently used. We find it using an index array.

We basically need to replace the page with

minimum index.

b) Replace the found page with the current page.

c) Increment page faults.

d) Update index of current page.

2. Return page faults.

PROGRAM

```
#include<stdio.h>

void main(){
   int n,i,j,req[10],qb = 0,cap,c = 0,set[5],pfault,flag = 0,val;
   printf("Enter the number of requests: ");
   scanf("%d",&n);
   printf("Enter the request sequence: ");
   for(i=0;i<n;i++){
       scanf("%d",&req[i]);
   }
   printf("Enter the capacity: ");
   scanf("%d",&cap);
   for(i=0;i<n;i++){
       if(c<cap){
           set[c] = req[i];
           c++;
           pfault++;
       }
       else{
           for(j=0;j<cap;j++){
               if(set[j] == req[i]){
                   flag = 1;
                   break;
               }
           }
           if(flag == 0){
               val = req[qb];
               qb++;
```

```
                for(j=0;j<cap;j++){
                    if(set[j] == val){
                        set[j] = req[i];
                        pfault++;
                        break;
                    }
                }
            }
        }
    }
    printf("Number of page faults: %d\n",pfault);
}
```

OUTPUT

```
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ gcc lru.c
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ ./a.out
Enter the number of requests: 5
Enter the request sequence: 7 0 1 2 0
Enter the capacity: 3
Number of page faults: 4
user@user-HP-Laptop-15q-ds0xxx:~/oslab$
```

c) LFU

ALGORITHM

1. Start

2. Read Number Of Pages And Frames

3.Read Each Page Value

4. Search For Page In The Frames

5. If Not Available Allocate Free Frame

6. If No Frames Is Free Replace The Page With The Page That Is Least Used

7.Print Page Number Of Page Faults

8.Stop

PROGRAM

```
#include <stdio.h>
int main(){
   int f, p;
   int pages[50], frame[10], hit = 0, count[50], time[50];
   int i, j, page, flag, least, minTime, temp;
   printf("Enter no of frames : ");
   scanf("%d", &f);
   printf("Enter no of pages : ");
   scanf("%d", &p);
   for (i = 0; i < f; i++){
       frame[i] = -1;
   }
   for (i = 0; i < 50; i++){
       count[i] = 0;
   }
   printf("Enter page no : \n");
   for (i = 0; i < p; i++){
       scanf("%d", &pages[i]);
   }
   printf("\n");
   for (i = 0; i < p; i++){
       count[pages[i]]++;
       time[pages[i]] = i;
       flag = 1;
       least = frame[0];
       for (j = 0; j < f; j++){
           if (frame[j] == -1 || frame[j] == pages[i]){
               if (frame[j] != -1){
                   hit++;
               }
               flag = 0;

               frame[j] = pages[i];
               break;
           }
           if (count[least] > count[frame[j]]){
               least = frame[j];
           }
```

```
        }
        if (flag){
            minTime = 50;
            for (j = 0; j < f; j++){
                if (count[frame[j]] == count[least] && time[frame[j]] <
minTime){
                    temp = j;
                    minTime = time[frame[j]];
                }
            }
            count[frame[temp]] = 0;
            frame[temp] = pages[i];
        }
        for (j = 0; j < f; j++){
            printf("%d ", frame[j]);
        }
        printf("\n");
    }
    printf("Page hit = %d\n", hit);
    return 0;
}
```

OUTPUT

```
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ ./a.out
Enter no of frames : 3
Enter no of pages : 5
Enter page no :
7 0 1 2 0

7 -1 -1
7 0 -1
7 0 1
2 0 1
2 0 1
Page hit = 1
```

Result:

Program finished and output obtained successfully.

SIMULATE CONTIGUOUS MEMORY ALLOCATION TECHNIQUES(worst-fit, best-fit, first-fit) USING C

Experiment No: 16

Date: 02/08/2022

AIM: To simulate the following contiguous memory allocation techniques.

a) Worst-fit b) Best-fit c) First-fit

a) Worst-fit

ALGORITHM

1- Input memory blocks and processes with sizes.

2- Initialise all memory blocks as free.

3- Start by picking each process and find the

maximum block size that can be assigned to

current process i.e., find max(bockSize[1],

blockSize[2],.....blockSize[n]) >

processSize[current], if found then assign

to the current process.

5- If not then leave that process and keep checking

the further processes.

PROGRAM

```
#include<stdio.h>
void main(){
   int n,i,proc[10],m,mem[10],alloc[10],idx,j;
   printf("Enter the number of processes: ");
   scanf("%d",&n);
   printf("Enter the size of the processes: ");
   for(i=0;i<n;i++){
       scanf("%d",&proc[i]);
   }
   printf("Enter the number of memory blocks: ");
   scanf("%d",&m);
   printf("Enter the memory blocks: ");
   for(i=0;i<m;i++){
       scanf("%d",&mem[i]);
   }
   for(i=0;i<n;i++){
       alloc[i] = -1;
   }
   for(i=0;i<n;i++){
       idx = -1;
       for(j=0;j<m;j++){
           if(mem[j] >= proc[i]){
               if(idx == -1)
                   idx = j;
               else if(mem[idx] < mem[j]){
                   idx = j;
               }
           }
       }
       if(idx != -1){
           alloc[i] = idx;
           mem[idx] -= proc[i];
       }}
   printf("\nProcess No.\tProcess Size\tBlock no.\n");
   for (i=0;i< n;i++){
       printf("   %d\t\t%d\t\t",i+1,proc[i]);
       if (alloc[i] != -1)
           printf("%d\n",alloc[i] + 1);
       else
```

```
            printf("Not Allocated\n");
    }}
```

OUTPUT

```
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ gcc wf.c
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ ./a.out
Enter the number of processes: 4
Enter the size of the processes: 212 417 112 426
Enter the number of memory blocks: 5
Enter the memory blocks: 100 500 200 300 600

Process No.     Process Size    Block no.
   1            212             5
   2            417             2
   3            112             5
   4            426             Not Allocated
```

b) Best-fit

ALGORITHM

1- Input memory blocks and processes with sizes.

2- Initialise all memory blocks as free.

3- Start by picking each process and find the minimum block size that can be assigned to current process i.e., find min(bockSize[1], blockSize[2],.....blockSize[n]) > processSize[current], if found then assign to the current process.

5- If not then leave that process and keep checking the further processes.

PROGRAM

```
#include<stdio.h>

void main(){
   int n,i,proc[10],m,mem[10],alloc[10],idx,j;
   printf("Enter the number of processes: ");
   scanf("%d",&n);
   printf("Enter the size of the processes: ");
   for(i=0;i<n;i++){
       scanf("%d",&proc[i]);
   }
   printf("Enter the number of memory blocks: ");
   scanf("%d",&m);
   printf("Enter the memory blocks: ");
   for(i=0;i<m;i++){
       scanf("%d",&mem[i]);
   }
   for(i=0;i<n;i++){
       alloc[i] = -1;
   }
   for(i=0;i<n;i++){
       idx = -1;
       for(j=0;j<m;j++){
           if(mem[j] >= proc[i]){
               if(idx == -1)
                   idx = j;
               else if(mem[idx] > mem[j]){
                   idx = j;
               }
           }
       }
       if(idx != -1){
           alloc[i] = idx;
           mem[idx] -= proc[i];
       }}
   printf("\nProcess No.\tProcess Size\tBlock no.\n");
   for (i=0;i< n;i++){
       printf("   %d\t\t%d\t\t",i+1,proc[i]);
       if (alloc[i] != -1)
           printf("%d\n",alloc[i] + 1);
       else
```

```
        printf("Not Allocated\n");
  }}
```

OUTPUT

```
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ ./a.out
Enter the number of processes: 4
Enter the size of the processes: 212 417 112 426
Enter the number of memory blocks: 5
Enter the memory blocks: 100 500 200 300 600

Process No.     Process Size    Block no.
   1            212             4
   2            417             2
   3            112             3
   4            426             5
user@user-HP-Laptop-15q-ds0xxx:~/oslab$
```

c) First-fit

ALGORITHM

1- Input memory blocks with size and processes with size.

2- Initialise all memory blocks as free.

3- Start by picking each process and check if it can
   be assigned to the current block.

4- If size-of-process <= size-of-block if yes then
   assign and check for the next process.

5- If not then keep checking the further blocks.

PROGRAM

```
#include<stdio.h>
void main(){
   int n,i,proc[10],m,mem[10],alloc[10],j;
   printf("Enter the number of processes: ");
   scanf("%d",&n);
   printf("Enter the size of the processes: ");
   for(i=0;i<n;i++){
```

```
        scanf("%d",&proc[i]);
    }
    printf("Enter the number of memory blocks: ");
    scanf("%d",&m);
    printf("Enter the memory blocks: ");
    for(i=0;i<m;i++){
        scanf("%d",&mem[i]);
    }
    for(i=0;i<n;i++){
        alloc[i] = -1;
    }
    for(i=0;i<n;i++){
        for(j=0;j<m;j++){
            if(mem[j] >= proc[i]){
                alloc[i] = j;
                mem[j] -= proc[i];
            }
        }
    }
    printf("\nProcess No.\tProcess Size\tBlock no.\n");
    for (i=0;i< n;i++){
        printf("   %d\t\t%d\t\t",i+1,proc[i]);
        if (alloc[i] != -1)
            printf("%d\n",alloc[i] + 1);
        else
            printf("Not Allocated\n");
    }
}
```

OUTPUT

```
user@user-HP-Laptop-15q-ds0xxx:~/oslab$ ./a.out
Enter the number of processes: 4
Enter the size of the processes: 212 417 112 426
Enter the number of memory blocks: 5
Enter the memory blocks: 100 500 200 300 600

Process No.     Process Size    Block no.
   1            212             5
   2            417             Not Allocated
   3            112             5
   4            426             Not Allocated
```

Result

Program finished and output obtained successfully.