

LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING

(AUTONOMOUS)

L.B. REDDY NAGAR, MYLAVARAM, KRISHNA DIST., A.P.-521 230.

DEPARTMENT OF INFORMATION TECHNOLOGY



Operating System and Linux Internals Lab (20IT54) B.TECH VI SEMESTER R20

G.Rajendra

Cycle 1:

Aim: Learn some of the basic concepts in OS with the help of Linux commands. Commands: ps, kill, killall, ls, ln, readlink, cp, rm, vi editor, grep, find, who, cat, who.

ps:

The **ps command** is used to view currently running processes on the system. It helps us to determine which process is doing what in our system, how much memory it is using, how much CPU space it occupies, user ID, command name, etc .

Syntax:

ps

output:

```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~$ ps  
  PID TTY          TIME CMD  
 6647 pts/1    00:00:00 bash  
 6706 pts/1    00:00:00 ps  
sssit@JavaTpoint:~$
```

Options

Option

ps -ef / -aux

ps -ax

ps -u <username>

ps -C <command>

ps -p <PID>

ps -ppid <PPID>

pstree

ps -L

ps --sort pmem

ps -eo

ps -U root -u root u

Function

List currently running process in full format

List currently running process

List process for specific user

List process for given command

List process with given PID

List process with given ppid

Show process in hierarchy

List all threads for a particular process

Find memory leak

Show security information

Show process running by root

Kill:

On Linux, the "**kill**" command is used to send a signal to a process, which can be used to kill the process. The signal can be specified as a signal number or as a signal name, and the default signal is the **TERM** signal, which terminates the process. In this article, we'll explore the different options and usage of the "**kill**" command, including how to use it to kill specific processes and how to use it in combination with other commands.

Syntax:

\$ kill [signal] pid

Example:

To stop a process with the **PID** of **1234**, you would enter the following command –

\$ kill -STOP 1234

To continue the process, you would use the following command –

\$ kill -CONT 1234

Killall:

The **killall** command cancels all processes that you started, except those producing the **killall** process. This command provides a convenient means of canceling all processes created by the shell that you control. When started by a root user, the **killall** command cancels all cancellable processes except those processes that started it. If several Signals are specified, only the last one is effective.

If no signal is specified, the **killall** command sends a **SIGKILL** signal.

Syntax

killall [-] [-Signal]

Examples

1. To stop all background processes that have started, enter:

```
killall
```

This sends all background processes the **kill** signal 9 (also called the **SIGKILL** signal).

2. To stop all background processes, giving them a chance to clean up, enter:

```
killall -
```

This sends signal 15, the **SIGTERM** signal; waits 30 seconds, and then sends signal 9, the **SIGKILL** signal.

3. To send a specific signal to the background processes, enter:

```
killall -2
```

This sends signal 2, the **SIGINT** signal, to the background processes.

ls:

The **ls** is the list command in Linux. It will show the full list or content of your directory. Just type **ls** and press the enter key. The whole content will be shown.

ls is a command used to list computer directories and files in Unix-like and Unix operating systems. It is developed by the Single Unix Specification and POSIX.

Syntax:

ls

output:

```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~$ pwd  
/home/sssit  
sssit@JavaTpoint:~$ ls  
Desktop  Downloads  Music      Public      Videos  
Documents examples.desktop  Pictures  Templates  
sssit@JavaTpoint:~$
```

ls command options

ls option	Description
<u>ls -a</u>	In Linux, hidden files start with . (dot) symbol and they are not visible in the regular directory. The (ls -a) command will enlist the whole list of the current directory including the hidden files.
<u>ls -l</u>	It will show the list in a long list format.
ls -lh	This command will show you the file sizes in human readable format. Size of the file is very difficult to read when displayed in terms of byte. The (ls -lh)command will give you the data in terms of Mb, Gb, Tb, etc.
ls -lhS	If you want to display your files in descending order (highest at the top) according to their size, then you can use (ls -lhS) command.
<u>ls -l - -block-size=[SIZE]</u>	It is used to display the files in a specific size format. Here, in [SIZE] you can assign size according to your requirement.
<u>ls -d */</u>	It is used to display only subdirectories.
<u>ls -g or ls -lG</u>	With this you can exclude column of group information and owner.
ls -n	It is used to print group ID and owner ID instead of their names.
<u>ls --color=[VALUE]</u>	This command is used to print list as colored or discolored.
ls -li	This command prints the index number if file is in the first column.
ls -p	It is used to identify the directory easily by marking the directories with a slash (/) line sign.
ls -r	It is used to print the list in reverse order.
ls -R	It will display the content of the sub-directories also.
ls -lX	It will group the files with same extensions together in the list.
ls -lt	It will sort the list by displaying recently modified file at top.
<u>ls ~</u>	It gives the contents of home directory.
<u>ls ./</u>	It give the contents of parent directory.
ls --version	It checks the version of ls command.

ln:

ln - make links between files

In the 1st form, create a link to TARGET with the name LINK_NAME. In the 2nd form, create a link to TARGET in the current directory. In the 3rd and 4th forms, create links to each TARGET in DIRECTORY. Create hard links by default, symbolic links with **--symbolic**. When creating hard links, each TARGET must exist.

options :

Tag	Description
--backup[=CONTROL]	
	make a backup of each existing destination file
-b	like --backup but does not accept an argument
-d, -F, --directory	
	allow the superuser to attempt to hard link directories (note: will probably fail due to system restrictions, even for the superuser)
-f, --force	
	remove existing destination files
-n, --no-dereference	
	treat destination that is a symlink to a directory as if it were a normal file
-i, --interactive	
	prompt whether to remove destinations
-s, --symbolic	
	make symbolic links instead of hard links
-S, --suffix=SUFFIX	
	override the usual backup suffix
-t, --target-directory=DIRECTORY	
	specify the DIRECTORY in which to create the links
-T, --no-target-directory	
	treat LINK_NAME as a normal file
-v, --verbose	
	print name of each file before linking
--help	display this help and exit
--version	
	output version information and exit
The backup suffix is '~', unless set with --suffix or SIMPLE_BACKUP_SUFFIX. The version control method may be selected via the --backup option or through the VERSION_CONTROL environment variable. Here are the values:	
none, off	
	never make backups (even if --backup is given)
numbered, t	
	make numbered backups
existing, nil	
	numbered if numbered backups exist, simple otherwise
simple, never	
	always make simple backups

Readlink:

readlink command in Linux is used to print resolved symbolic links or canonical file names. In simple words whenever we have a symbolic link and we want to know what path it represents. Then, in that case, the *readlink* command comes into play to show the actual path of the symbolic link.

Syntax:

readlink [OPTION]... FILE...

Example: It will print the print resolved symbolic links or canonical file names of the symbolic link passed with the command as shown below.

```
algoscale@algoscale-Lenovo-ideapad-330-15IKB:~/bin$ ls
desk desk1 desk2 desk4 desk5 desk6
algoscale@algoscale-Lenovo-ideapad-330-15IKB:~/bin$ readlink desk
/home/algoscale/Desktop
```

cp:

cp command copies files (or, optionally, directories). The copy is completely independent of the original. You can either copy one file to another, or copy arbitrarily many files to a destination directory.

In the first format, when two file names are given, cp command copies SOURCE file to DEST file.

In the second format copy multiple SOURCE(s) to a DIRECTORY.

Options

-a, --archive

same as -dR --preserve=all

--attributes-only

don't copy the file data, just the attributes

--backup[=CONTROL]

make a backup of each existing destination file

-b like --backup but does not accept an argument

--copy-contents

copy contents of special files when recursive

-d same as --no-dereference --preserve=links

-f, --force

if an existing destination file cannot be opened, remove it and try again (this option is ignored when the -n option is also used)

-i, --interactive

prompt before overwrite (overrides a previous -n option)

-H follow command-line symbolic links in SOURCE

-l, --link

hard link files instead of copying

-L, --dereference

always follow symbolic links in SOURCE

-n, --no-clobber

do not overwrite an existing file (overrides a previous -i option)

-P, --no-dereference

never follow symbolic links in SOURCE

- p same as --preserve=mode,ownership,timestamps
- preserve[=ATTR_LIST]
preserve the specified attributes (default: mode,ownership,timestamps), if possible additional attributes: context, links, xattr, all
- no-preserve=ATTR_LIST
don't preserve the specified attributes
- parents
use full source file name under DIRECTORY
- R, -r, --recursive
copy directories recursively
- reflink[=WHEN]
control clone/CoW copies. See below
- remove-destination
remove each existing destination file before attempting to open it (contrast with --force)
- sparse=WHEN
control creation of sparse files. See below
- strip-trailing-slashes
remove any trailing slashes from each SOURCE argument
- s, --symbolic-link
make symbolic links instead of copying
- S, --suffix=SUFFIX
override the usual backup suffix
- t, --target-directory=DIRECTORY
copy all SOURCE arguments into DIRECTORY
- T, --no-target-directory
treat DEST as a normal file
- u, --update
copy only when the SOURCE file is newer than the destination file or when the destination file is missing
- v, --verbose
explain what is being done
- x, --one-file-system
stay on this file system
- Z set SELinux security context of destination file to default type
- context[=CTX]
like -Z, or if CTX is specified then set the SELinux or SMACK security context to CTX

--help display this help and exit

--version

output version information and exit

rm:

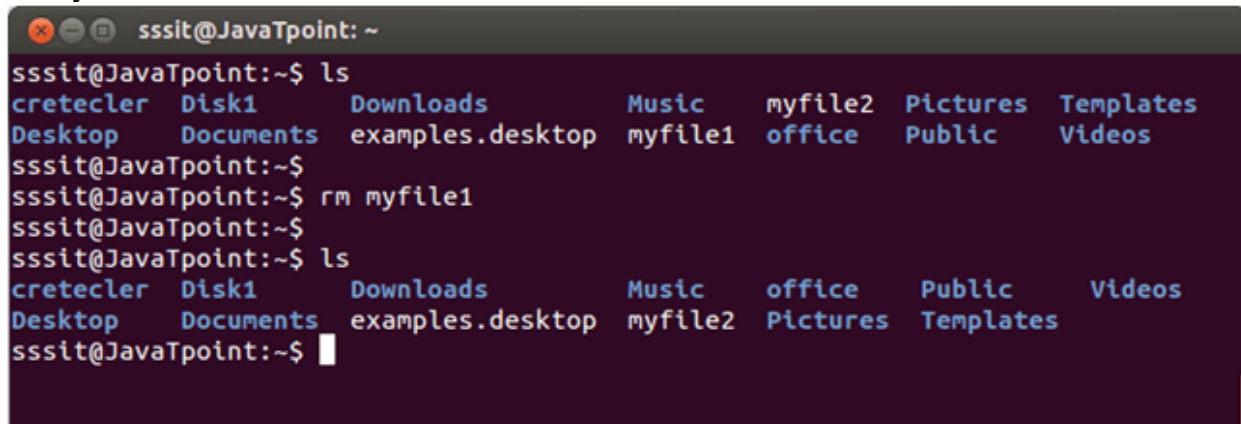
The 'rm' means remove. This command is used to remove a file. The command line doesn't have a recycle bin or trash unlike other GUI's to recover the files. Hence, be very much careful while using this command. Once you have deleted a file, it is removed permanently.

Syntax:

rm <filename>

Example:

rm myfile1

A terminal window titled 'sssit@JavaTpoint: ~' showing the execution of the 'rm' command. The user runs 'ls' and lists files: 'cretecler', 'Disk1', 'Downloads', 'Music', 'myfile2', 'Pictures', 'Templates', 'Desktop', 'Documents', 'examples.desktop', 'myfile1', 'office', 'Public', 'Videos'. Then they run 'rm myfile1'. After another 'ls', 'myfile1' is no longer in the list, and 'office' has moved to 'Pictures'.

```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~$ ls  
cretecler  Disk1      Downloads  Music      myfile2    Pictures    Templates  
Desktop    Documents  examples.desktop  myfile1    office     Public      Videos  
sssit@JavaTpoint:~$  
sssit@JavaTpoint:~$ rm myfile1  
sssit@JavaTpoint:~$  
sssit@JavaTpoint:~$ ls  
cretecler  Disk1      Downloads  Music      office     Public      Videos  
Desktop    Documents  examples.desktop  myfile2    Pictures    Templates  
sssit@JavaTpoint:~$
```

Options:

Option

Description

rm *extension

Used to delete files having same extension.

rm -r or R

To delete a directory recursively.

rm -i

Remove a file interactively.

rm -rf

Remove a directory forcefully.

vi editor:

The vi editor is elaborated as **visual** editor. It is installed in every Unix system. In other words, it is available in all Linux distros. It is user-friendly and works same on different distros and platforms. It is a very powerful application. An improved version of vi editor is **vim**.

The vi editor has two modes:

- **Command Mode:** In command mode, actions are taken on the file. The vi editor starts in command mode. Here, the typed words will act as commands in vi editor. To pass a command, you need to be in command mode.
- **Insert Mode:** In insert mode, entered text will be inserted into the file. The **Esc** key will take you to the command mode from insert mode.

Using vi

The vi editor tool is an interactive tool as it displays changes made in the file on the screen while you edit the file.

In vi editor you can insert, edit or remove a word as cursor moves throughout the file.

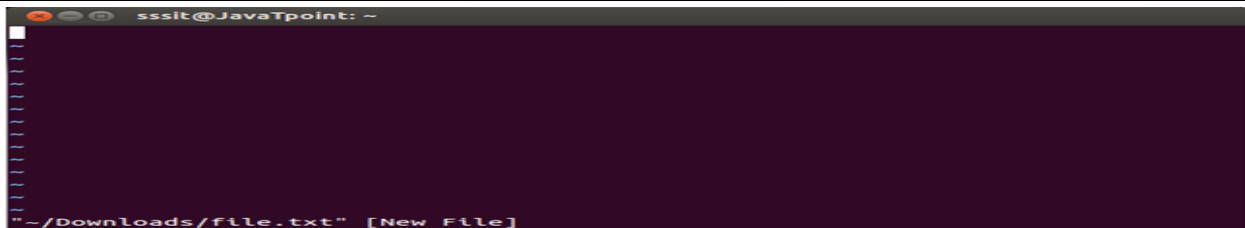
Commands are specified for each function like to delete it's x or dd.

syntax:

vi <fileName>

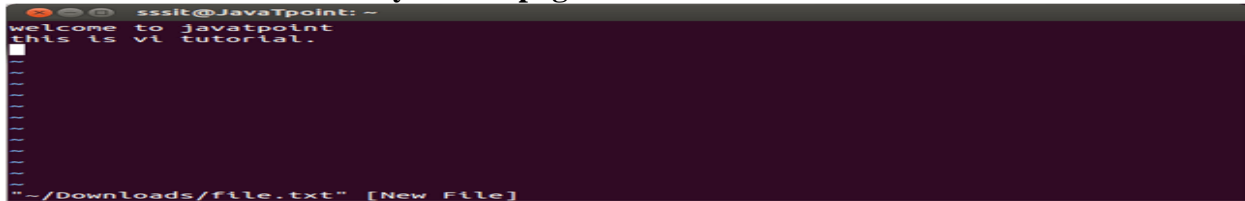
Command mode

This is what you'll see when you'll press enter after the above command. If you'll start typing, nothing will appear as you are in command mode. By default vi opens in command mode.

A terminal window with a dark purple background. The prompt is 'sssit@JavaTpoint: ~'. Below the prompt, there are several lines of vertical bars representing a file list. The last line is highlighted in blue and reads: "~/Downloads/file.txt" [New File].

Insert mode

To move to the insert mode press **i**. Although, there are other commands also to move to insert mode which we'll study in next page.

A terminal window with a dark purple background. The prompt is 'sssit@JavaTpoint: ~'. Below the prompt, there are several lines of vertical bars. The last line is highlighted in blue and contains the text: 'welcome to javatpoint' and 'this is vi tutorial.'.

To save and quit

You can save and quit vi editor from command mode. Before writing save or quit command you have to press colon (:). Colon allows you to give instructions to vi.

exit vi table:

Commands	Action
:wq	Save and quit
:w	Save
:q	Quit
:w fname	Save as fname
ZZ	Save and quit
:q!	Quit discarding changes made
:w!	Save (and write to non-writable file)

grep:

The 'grep' command stands for "**global regular expression print**". grep command filters the content of a file which makes our search easy. It is a command-line utility to search plain-text data groups for lines that are the same as a regular expression. The name "**grep**" comes from the command, i.e., ed, which contains the same effect. Originally, grep was designed for the Unix operating system, but it became available for every Unix-like system later and a few others like OS 9.

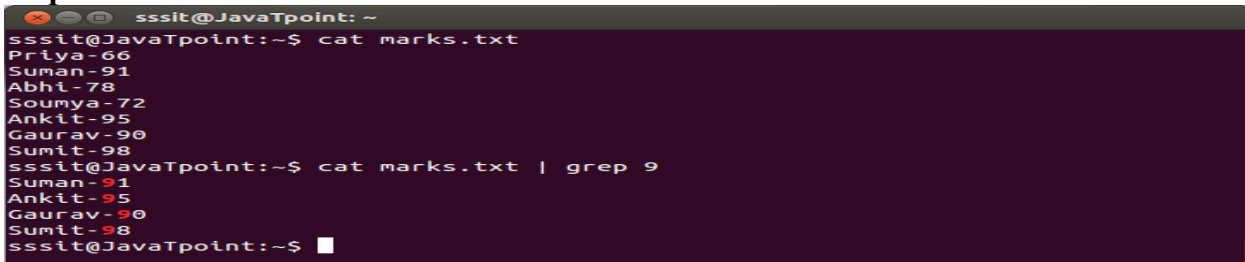
Syntax:

command | grep <searchWord>

Example:

cat marks.txt | grep 9

output:

A terminal window with a dark purple background. The prompt is 'sssit@JavaTpoint: ~'. The user enters 'cat marks.txt' and the output is: Priya-66, Suman-91, Abhi-78, Soumya-72, Ankit-95, Gaurav-90, Sumit-98. Then the user enters 'cat marks.txt | grep 9' and the output is: Suman-91, Ankit-95, Gaurav-90, Sumit-98.

grep without pipe

It can be used without pipe also.

Syntax:

grep <searchWord> <file name>

Example:

grep 9 marks.txt


```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~$ grep 9 marks.txt  
Suman-91  
Ankit-95  
Gaurav-90  
Sumit-98  
sssit@JavaTpoint:~$
```

Find:

The find command helps us to find a particular file within a directory. It is used to find the list of files for the various conditions like permission, user ownership, modification, date/time, size, and more.

find <location> <comparison-criteria> <search-term>

example:

find . -name "*.txt"

output:

```
sssit@JavaTpoint: ~/Downloads  
sssit@JavaTpoint:~/Downloads$ find . -name "*.txt"  
./linux index.txt  
./msg.txt  
./format.txt  
./dupli.txt  
./marks.txt  
sssit@JavaTpoint:~/Downloads$
```

who:

The Linux "who" command lets you display the users currently logged in to your UNIX or Linux operating system.

Syntax

who

```
akash@akash-VirtualBox: ~  
akash@akash-VirtualBox:~$ who  
akash :0 2021-03-26 23:04 (:0)  
akash@akash-VirtualBox:~$
```

To display all details of currently logged in users-

With this command's help, one sees all the details of every user logged in to the current system. The syntax of this command is the same except the additional option "-a", as we can see in the given syntax:

Syntax

who -a

Output

```
akash@akash-VirtualBox: ~  
akash@akash-VirtualBox:~$ who -a  
system boot 2021-03-26 23:01  
run-level 5 2021-03-26 23:03  
akash ? :0 2021-03-26 23:04 ? 1382 (:0)  
akash@akash-VirtualBox:~$
```

cat:

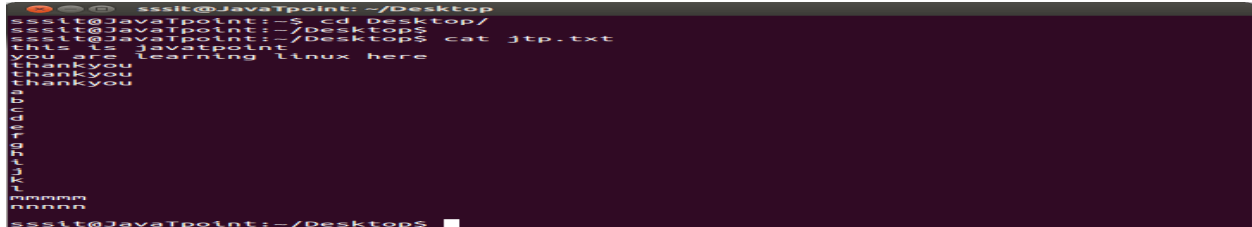
The 'cat' command is the most universal and powerful tool. It is considered to be one of the most frequently used commands. It can be used to display the content of a file, copy content from one file to another, concatenate the contents of multiple files, display the line number, display \$ at the end of the line, etc.

Options in the cat Command

- **--show-all, -A:** It is the same as -vET.
- **--number-nonblank, -b:** It shows the total non-empty output lines. Also, it overrides -n.
- **-e:** It is the same as -vE.
- **--show-ends, -E:** It shows the \$ symbol at the completion of all lines.

- cat <**fileName**>

```
cat jtp.txt
```



Option	Function
cat > [fileName]	To create a file.
cat [oldfile] > [newfile]	To copy content from older to new file.
cat [file1 file2 and so on] > [new file name]	To concatenate contents of multiple files into one.
cat -n/cat -b [fileName]	To display line numbers.
cat -e [fileName]	To display \$ character at the end of each line.
cat [fileName] <<EOF	Used as page end marker.

Cycle 2:

Aim: Introduce system calls in the Linux OS with the help of some basic system calls such as fork, exec, sleep, wait, etc.

(a) fork() system call

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
int main(int argc, char **argv) {
    pid_t pid = fork();
    if (pid==0) {
        printf("This is the Child process and pid is: %d\n",getpid());
        exit(0);
    } else if (pid > 0) {
        printf("This is the Parent process and pid is: %d\n",getpid());
    } else {
        printf("Error while forking\n");
        exit(EXIT_FAILURE);
    }
    return 0;
}
```

Compilation: gcc fork.c

Output:

\$./a.out

This is the Parent process and pid is: 69032

This is the Child process and pid is: 69033

(B)(i) C program to demonstrate working of wait()

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
int main()
{
    pid_t cpid;
    if (fork()== 0)
        exit(0);      /* terminate child */
    else
        cpid = wait(NULL); /* reaping parent */
    printf("Parent pid = %d\n", getpid());
    printf("Child pid = %d\n", cpid);
    return 0;
}
```

Output:

Parent pid = 12345678

Child pid = 89546848

(B)(ii) C program to demonstrate working of wait()

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

```
#include<sys/wait.h>
```

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

```
int main()
{
    if (fork()== 0)
        printf("HC: hello from child\n");
    else
    {
        printf("HP: hello from parent\n");
        wait(NULL);
        printf("CT: child has terminated\n");
    }

    printf("Bye\n");
    return 0;
}
```

Output: depend on environment

```
HC: hello from child
HP: hello from parent
CT: child has terminated
(or)
HP: hello from parent
HC: hello from child
CT: child has terminated
```

(C) Write a C program that illustrates the creation of child process using fork system call. One process finds sum of even series and other process finds sum of odd series.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
int main(){
int i,n,sum=0;
pid_t pid;
system("clear");
printf("Enter n value:");
scanf("%d",&n)
pid=fork();
if(pid==0)
{
printf("From child process\n");
for(i=1;i<n;i+=2)
{
printf("%d\\",i);
sum+=i;
}
printf("Odd sum:%d\\n",sum);
}
else
{
printf("From process\n");
for(i=0;i<n;i+=2)
{
printf("%d\\",i);
sum+=i;
}
printf("Even sum:%d\\n",sum);
}}
```

Output:

Enter n value:10

From Parent process

0	2	4	6	8	Even sum=20
---	---	---	---	---	-------------

From child process

1	3	5	7	9	Odd sum=25
---	---	---	---	---	------------

(D) `exec()` family of functions:

These functions are used to execute a file, and they replace the current process image with a new process image once they are called.

`execl()` receives the location of the executable file as its first argument. The next arguments will be available to the file when it's executed. The last argument has to be `NULL`:

```
int execl(const char *pathname, const char *arg, ..., NULL)
```

Let's look at an example. We need to make sure to include `unistd.h`:

```
#include <unistd.h>  
int main(void) {  
    char *file = "/usr/bin/echo";  
    char *arg1 = "Hello world!";  
  
    execl(file, file, arg1, NULL);  
  
    return 0;  
}
```

The command we are running is `echo` which is located at `/usr/bin/echo`. By convention, the first argument available to a program needs to be the program itself.

Output:

```
$ gcc execl.c
```

```
$ ./a.out
```

```
Hello world!
```

`echo` has successfully printed the output on the screen.

`execlp()`

`execlp()` is very similar to `execl()`. However, `execlp()` uses the `PATH` environment variable to look for the file. Therefore, the path to the executable file is not needed:

```
int execlp(const char *file, const char *arg, ..., NULL)Copy
```

Let's see an example:

```
#include <unistd.h>  
int main(void) {  
    char *file = "echo";  
    char *arg1 = "Hello world!";  
  
    execlp(file, file, arg1, NULL);  
    return 0;  
}
```

Output:

```
$ gcc execlp.c
```

```
$ ./a.out
```

```
Hello world!
```

Since `echo` is already located in the `PATH` environment variable, we didn't have to specify its location.

sleep() system call (Zombie Process):

```
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    // Fork returns process id
    // in parent process
    pid_t pid = fork();

    // Parent process
    if (pid > 0)
        sleep(50);
    // Child process
    else
        exit(0);
    return 0;
}
```

Output:

gcc zombie.c

./a.out

Process id=11577

Process id=11578

Develop C program to demonstrate Orphan Process

In the following code, parent finishes execution and exits while the child process is still executing and is called an orphan process now.

However, the orphan process is soon adopted by init process, once its parent process dies.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    // Create a child process
    int pid = fork();
    if (pid > 0)
        printf("in parent process");

    // Note that pid is 0 in child process
    // and negative if fork() fails
    else if (pid == 0)
    {
        sleep(30);
        printf("in child process");
    }
    return 0;
}
```

Output:

gcc orphan.c

./a.out

in parent process

in child process

Cycle 3:

(a) Write a shell script that accepts a file name, starting and ending line numbers as arguments and displays all the lines between the given line numbers.

//Program

```
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
```

Output:

```
cat sample
Hello
2nd Year Students
Welcome
to
Os Lab
Good Bye
$ chmod +x cycle3.sh
$ ./cycle3.sh
enter the filename
sample
enter the starting line number
1
enter the ending line number
3
Hello
2nd Year Students
Welcome
```

(b) Write a shell script to perform arithmetic operations of two numbers.

```
#!/bin/sh
# take two numbers from user
echo "Enter a value: "
read a
echo "Enter b value: "
read b
val=`expr $a + $b`
echo "a + b : $val"
val=`expr $a - $b`
echo "a - b : $val"
val=`expr $a \* $b`
echo "a * b : $val"
val=`expr $b / $a`
echo "b / a : $val"
val=`expr $b % $a`
echo "b % a : $val"
if [ $a == $b ]
then
    echo "a is equal to b"
fi
```



```
if [ $a != $b ]
then
    echo "a is not equal to b"
fi
```

Output:

```
Enter a value:
10
Enter b value:
5
a+b:15
a-b:5
a*b:50
b/a:0
b%a:5
a is not equal to b
```

(c) Write a shell script which counts the number of lines and words present in a given file.

```
#!/bin/sh
#shell script to count the no of lines, words and characters in the given file
echo "Enter a file name:"
read fn
echo "Number of Lines:"
wc -l $fn
echo "Number of Words:"
wc -w $fn
echo "Number of Characters:"
wc -c $fn
```

Output:

```
Enter a file name:
Samp1
Number of Lines:
3 Samp1
Number of Words:
8 Samp1
Number of Characters:
56 Samp1
```

Cycle-4

(a) Write a shell script that displays the list of all files in the given directory.

```
echo "Menu"
echo "1.Short format display \n"
echo "2.Long format display \n"
echo "3.Hidden files to display \n"
echo "Enter your choice:"
read ch
case $ch in
1) ls $a
;;
2) ls -l $a
;;
3) ls -la $a
;;
*) echo "Choice is not correct"
;;
esac
```

Output:

```
root@lbrcmca ~# bash fldisplay.sh
Menu
1.Short format display\n
2.Long format display\n
3.Hidden files to display\n
Enter your choice
1
07761F0055      errorfile      fprint      palindrome.sh      sample2.txt
1      factorial.sh      fprint.c      paste.txt      sample3.txt
anaconda-ks.cfg      factors.sh      fscanf      pig_1568243956467.log      sample.c
arithoperations.sh~      fact.sh      fscanf.c      pig_1568244573857.log      sample.txt
arith.sh      fibonacci.sh      heredoc1.sh      pipes2way      sort1.txt
arth.sh      fifoclient      heredoc.sh      pipes2way.c      sort2.txt
callingfork      fifoclient.c      ifthen.sh      prime1.sh      sumofdigits.sh
callingfork.c      fifoserver      implementls      prime.sh      swap.sh
catcommand      fifoserver.c      implementls.c      raju      swithcase.sh
catcommand.c      file2      implementsystemcalls      RAJU      test1.txt
cfd.sh      file4.txt      implementsystemcalls.c      reversenumber.sh      testfile
copy1.txt      file1cw.sh      insert.txt      root      test.sh
countblanks      fldisplay.sh      mca      root1      unsetvariable.sh
countblanks1      fork      mca1      root2      users
countblanks1.c      fork1      mulfiles.sh      root3      variable.sh
countblanks.c      fork1.c      multiplicationtable.sh      root5      vfork
countlwc.sh      fork2      mult.sh      root6      vfork.c
count.sh      fork2.c      mcommand      root7      zombie
create.txt      fork.c      mcommand.c      root8      zombie.c
emp      forkevenodd      MTFIFO      root9
empinfo      forkevenodd.c      my-script.sh      sample
empinfo.c      forkexec      orphan      sample1
emp.txt      forkexec.c      orphan.c      sample1.txt
root@lbrcmca ~# bash fldisplay.sh
Menu
1.Short format display\n
2.Long format display\n
3.Hidden files to display\n
Enter your choice
2
-rw-r--r-- 1 root root 245 Jul 24 14:09 prime1.sh
-rw-r--r-- 1 root root 347 Jul 24 14:15 prime.sh
-rw-r--r-- 1 root root 13 Jul 17 14:43 raju
drwxr-xr-x 2 root root 62 Sep 11 09:47 RAJU
-rw-r--r-- 1 root root 248 Jul 24 18:42 reversenumber.sh
drwxr-xr-x 2 root root 28 Jul 18 13:54 root
drwxr-xr-x 2 root root 35 Jul 18 18:28 root1
dr--x--t 2 root root 6 Jul 27 12:11 root2
drwxr-xr-x 2 root root 6 Jul 18 18:14 root3
dr--x--t 2 root root 38 Sep 16 09:52 root5
drwxr-xr-x 2 root root 6 Sep 11 09:41 root6
drwxr-xr-x 2 root root 63 Sep 11 09:43 root7
drwxr-xr-x 2 root root 59 Sep 11 09:45 root8
drwxr-xr-x 2 root root 48 Sep 16 09:53 root9
drwxr-xr-x 1 root root 8448 Jul 15 11:21 sample
-rw-r--r-- 1 root root 33 Jun 26 14:01 sample1
-rw-r--r-- 1 root root 37 Aug 13 14:36 sample1.txt
-rw-r--r-- 1 root root 25 Aug 22 15:55 sample2.txt
-rw-r--r-- 1 root root 28 Aug 22 15:38 sample3.txt
-rw-r--r-- 1 root root 71 Jun 24 11:58 sample.c
-rw-r--r-- 1 root root 35 Aug 22 15:27 sample.txt
-rw-r--r-- 1 root root 15 Jul 24 14:25 sort1.txt
-rw-r--r-- 1 root root 17 Jul 24 14:23 sort2.txt
-rw-r--r-- 1 root root 267 Jul 18 13:42 sumofdigits.sh
-rw-r--r-- 1 root root 281 Jul 9 16:24 swap.sh
-rw-r--r-- 1 root root 328 Jul 11 10:57 swithcase.sh
-rw-r--r-- 1 root root 48 Aug 22 14:41 test1.txt
-rw-r--r-- 1 root root 28 Jun 24 11:47 testfile
-rwxr-xr-x 1 root root 68 Jul 5 15:48 test.sh
-rw-r--r-- 1 root root 44 Jul 28 18:31 unsetvariable.sh
-rw-r--r-- 1 root root 14 Jul 17 13:23 users
-rw-r--r-- 1 root root 122 Jul 28 18:28 variable.sh
-rwxr-xr-x 1 root root 8688 Sep 11 18:28 vfork
```

```

-rw-r--r-- 1 root root 25 Aug 22 15:55 sample2.txt
-rw-r--r-- 1 root root 28 Aug 22 15:38 sample3.txt
-rw-r--r-- 1 root root 71 Jun 24 11:58 sample.c
-rw-r--r-- 1 root root 35 Aug 22 15:27 sample.txt
-rw-r--r-- 1 root root 15 Jul 24 14:25 sort1.txt
-rw-r--r-- 1 root root 17 Jul 24 14:23 sort2.txt
drwx----- 2 root root 88 Jun 11 12:14 .ssh
-rw-r--r-- 1 root root 267 Jul 18 13:42 sumofdigits.sh
-rw-r--r-- 1 root root 281 Jul 9 16:24 swap.sh
-rw-r--r-- 1 root root 328 Jul 11 18:57 swithcase.sh
-rw-r--r-- 1 root root 12288 Jul 11 12:27 swithcase.sh.swap
-rw----- 1 root root 12288 Jul 17 13:42 .swa
-rw----- 1 root root 12288 Jul 17 13:41 .swo
-rw----- 1 root root 12288 Jul 17 13:39 .swap
-rw-r--r-- 1 root root 129 Dec 29 2013 .tcshrc
-rw-r--r-- 1 root root 48 Aug 22 14:41 test1.txt
-rw-r--r-- 1 root root 28 Jun 24 11:47 testfile
-rwxr-xr-x 1 root root 68 Jul 5 15:48 test.sh
-rw----- 1 root root 12288 Jul 17 13:36 .test.txt.swap
-rw----- 1 root root 12288 Jul 17 13:36 .test.txt.swo
-rw-r--r-- 1 root root 44 Jul 28 18:31 unsetvariable.sh
-rw-r--r-- 1 root root 14 Jul 17 13:23 users
-rw-r--r-- 1 root root 122 Jul 28 18:28 variable.sh
-rwxr-xr-x 1 root root 8688 Sep 11 18:28 vfork
-rw-r--r-- 1 root root 612 Sep 11 18:25 vfork.c
-rw-r--r-- 1 root root 1886 Jun 25 15:04 .viminfo
-rwxr-xr-x 1 root root 8648 Sep 11 18:32 zombie
-rw-r--r-- 1 root root 583 Sep 11 18:38 zombie.c
root@lbrceemca ~]# bash fldisplay.sh
Menu
1.Short format display\n
2.Long format display\n
3.Hidden files to display\n
Enter your choice
1
choice is not correct
root@lbrceemca ~]# _

```

(b) Write a shell script to generate Fibonacci series.

```
#!/bin/sh
```

```
#shell script to generate Fibonacci series
```

```
clear
```

```
echo "How many no of terms to be generated?"
```

```
read n
```

```
x=0
```

```
y=1
```

```
i=2
```

```
echo "Fibonacci series up to $n terms:"
```

```
echo "$x"
```

```
echo "$y"
```

```
while [ $i -lt $n ]
```

```
do
```

```
    i=`expr $i + 1`
```

```
    z=`expr $x + $y`
```

```
    echo "$z"
```

```
    x=$y
```

```
    y=$z
```

```
done
```

Output:

```
How many no of terms to be generated?
```

```
5
```

```
Fibonacci series up to 5 terms:
```

```
0
```

```
1
```

```
1
```

```
2
```

```
3
```

(c) Write a shell script to print prime numbers in a given range.

```
#!/bin/bash
prime_1=0
echo "enter the range"
read n
echo " Prime number between 1 to $n is:"
echo "1"
echo "2"
for(( i=3;i<=n; ))
do
for(( j=i-1;j>=2; ))
do
if [ `expr $i % $j` -ne 0 ]
then
prime_1=1
else
prime_1=0
break
fi
j=`expr $j - 1`
done
if [ $prime_1 -eq 1 ]
then
echo $i
fi
i=`expr $i + 1`
done
```

Output:

```
enter the range
10
Prime number between 1 to 10 is:
1
2
3
5
7
```

Cycle 5:

(a) Write a shell script that deletes all lines containing a specified word in one or more files supplied as arguments to it.

```
echo enter file name
read file
echo enter word
read word
echo file before removing $word:
cat $file
grep -v -i $word $file > test
mv test $file
echo file after removing $word:
cat $file
```

Output:

```
cilab@ubuntu:~$ chmod +x cyc4.sh
cilab@ubuntu:~$ ./cyc4.sh
enter file name : sample
enter word
Hello
file before removing Hello:
Hello
2nd Year Students
to
Os Lab
Good Bye
file after removing Hello:
2nd Year Students
to
Os Lab
Good Bye
```

(b) Write a shell script to check whether the given number is palindrome or not.

```
echo "Enter a number:"
read n
num=$n
rev=0
while [ $n -gt 0 ]
do
a=`expr $n % 10`
n=`expr $n / 10`
rev=`expr $rev \* 10 + $a`
done
echo $rev
if [ sum -eq $rev ]
then
echo "Number is Palindrome"
else
echo "Number is not Palindrome"
fi
```

Output:

```
Enter a number:
121
121
Number is Palindrome
```

Cycle: 6

(a). Write a C program to illustrate the mkdir(), opendir(), readdir(), closedir() and rmdir() system calls.

```
#include<stdio.h>
#include<fcntl.h>
#include<dirent.h>
main()
{
char d[10];
int c,op;
DIR *e;
struct dirent *sd;
printf("**menu**\n1.create dir\n2.remove dir\n 3.read dir\n enter ur choice");
scanf("%d",&op);
switch(op)
{
case 1:
printf("enter dir name\n");
scanf("%s",&d);
c=mkdir(d,777);
if(c==1)
printf("dir is not created");
else
printf("dir is created");
break;

case 2: printf("enter dir name\n");
scanf("%s",&d);
c=rmdir(d);
if(c==1)
printf("dir is not removed");
else
printf("dir is removed");
break;
case 3:
printf("enter dir name to open");
scanf("%s",&d);
e=opendir(d);
if(e==NULL)
printf("dir does not exist");
else
{
printf("dir exist\n");
while((sd=readdir(e))!=NULL)
printf("%s\t",sd->d_name);
}
closedir(e);
break;
}
}
```

Output:

****menu****

1.create dir
2.remove dir
3.read dir
enter ur choice1
enter dir name
root5
dir is created

****menu****

1.create dir
2.remove dir
3.read dir
enter ur choice2
enter dir name
root4
dir is removed

****menu****

1.create dir
2.remove dir
3.read dir
enter ur choice3
enter dir name to open
root1
dir exist:
sample test.sh

****menu****

1.create dir
2.remove dir
3.read dir
enter ur choice4

(b) Write a shell script to demonstrate the usage getwd() System call.

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

int main() {
    char cwd[PATH_MAX];
    if (getcwd(cwd, sizeof(cwd)) != NULL) {
        printf("Current working dir: %s\n", cwd);
    } else {
        perror("getcwd() error");
        return 1;
    }
    return 0;
}
```

Output:

```
gcc getwd.c
./a.out
Current working dir:/home/cilab
```


Cycle-7:

Write a program to simulate the following CPU scheduling algorithms to find turnaround time and waiting time. i) Round Robin ii) FCFS

Round Robin Scheduling

```
#include<stdio.h>
void main()
{
    int i, NOP, sum=0, count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
    float avg_wt, avg_tat;
    printf(" Total number of process in the system: ");
    scanf("%d", &NOP);
    y = NOP;

    for(i=0; i<NOP; i++)
    {
        printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
        printf(" Arrival time is: \t");
        scanf("%d", &at[i]);
        printf(" \nBurst time is: \t");
        scanf("%d", &bt[i]);
        temp[i] = bt[i];
    }
    printf("Enter the Time Quantum for the process: \t");
    scanf("%d", &quant);
    printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
    for(sum=0, i = 0; y!=0; )
    {
        if(temp[i] <= quant && temp[i] > 0)
        {
            sum = sum + temp[i];
            temp[i] = 0;
            count=1;
        }
        else if(temp[i] > 0)
        {
            temp[i] = temp[i] - quant;
            sum = sum + quant;
        }
        if(temp[i]==0 && count==1)
        {
            y--;
            printf("\nProcess No[%d] \t\t %d\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);
            wt = wt+sum-at[i]-bt[i];
            tat = tat+sum-at[i];
            count =0;
        }
        if(i==NOP-1)
        {
            i=0;
        }
        else if(at[i+1]<=sum)
        {
            i++;
        }
    }
}
```

```

}
else
{
    i=0;
}
}
avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
}

```

Output:

Total number of process in the system: 6

Enter the Arrival and Burst time of the Process[1]

Arrival time is: 0

Burst time is: 8

Enter the Arrival and Burst time of the Process[2]

Arrival time is: 1

Burst time is: 4

Enter the Arrival and Burst time of the Process[3]

Arrival time is: 2

Burst time is: 2

Enter the Arrival and Burst time of the Process[4]

Arrival time is: 3

Burst time is: 1

Enter the Arrival and Burst time of the Process[5]

Arrival time is: 4

Burst time is: 3

Enter the Arrival and Burst time of the Process[6]

Arrival time is: 5

Burst time is: 2

Enter the Time Quantum for the process: 2

Process No	Burst Time	TAT	Waiting Time
Process No[3]	2	4	2
Process No[4]	1	4	3
Process No[6]	2	6	4
Process No[2]	4	14	10
Process No[5]	3	12	9
Process No[1]	8	20	12

Average Turn Around Time: 6.666667

Average Waiting Time: 10.000000

First Come First Serve Scheduling

```
#include<stdio.h>
main()
{
int bt[20], wt[20], tat[20], i, n;
float wtavg, tatavg;
printf("\nEnter the number of processes:");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\nEnter Burst Time for Process %d:", i);
scanf("%d", &bt[i]);
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\tPROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);
printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
}
```

Output:

Enter the number of processes:4

Enter Burst Time for Process 0:3

Enter Burst Time for Process 1:7

Enter Burst Time for Process 2:10

Enter Burst Time for Process 3:1

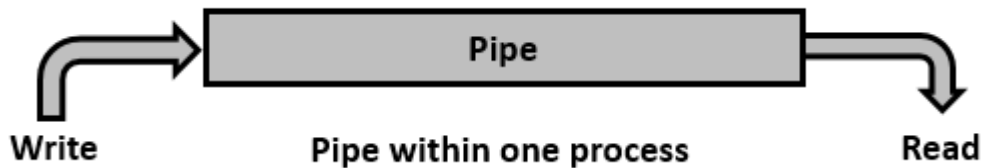
PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
P0	3	0	3
P1	7	3	10
P2	10	10	20
P3	1	20	21

Average Waiting Time -- 8.250000

Average Turnaround Time -- 13.500000

Cycle-8:

(a) Write a C Program to write and read two messages using pipe.



Algorithm

Step 1 – Create a pipe.

Step 2 – Send a message to the pipe.

Step 3 – Retrieve the message from the pipe and write it to the standard output.

Step 4 – Send another message to the pipe.

Step 5 – Retrieve the message from the pipe and write it to the standard output.

Note – Retrieving messages can also be done after sending all messages.

Source Code:

```
#include<stdio.h>
#include<unistd.h>
int main() {
    int pipefds[2];
    int returnstatus;
    char writemessages[2][20]={"Hi", "Hello"};
    char readmessage[20];
    returnstatus = pipe(pipefds);

    if (returnstatus == -1)
    {
        printf("Unable to create pipe\n");
        return 1;
    }
    printf("Writing to pipe - Message 1 is %s\n", writemessages[0]);
    write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
    read(pipefds[0], readmessage, sizeof(readmessage));
    printf("Reading from pipe – Message 1 is %s\n", readmessage);
    printf("Writing to pipe - Message 2 is %s\n", writemessages[1]);
    write(pipefds[1], writemessages[1], sizeof(writemessages[1]));
    read(pipefds[0], readmessage, sizeof(readmessage));
    printf("Reading from pipe – Message 2 is %s\n", readmessage);
    return 0;
}
```

Output:

Compilation

```
gcc -o simplepipe simplepipe.c
```

Execution/Output

```
Writing to pipe - Message 1 is Hi
Reading from pipe – Message 1 is Hi
Writing to pipe - Message 2 is Hello
Reading from pipe – Message 2 is Hello
```

(b) **Write C Program** to write and read two messages through the pipe using the parent and the child processes.

Algorithm

Step 1 – Create a pipe.

Step 2 – Create a child process.

Step 3 – Parent process writes to the pipe.

Step 4 – Child process retrieves the message from the pipe and writes it to the standard output.

Step 5 – Repeat step 3 and step 4 once again.

Source Code:

```
#include<stdio.h>
#include<unistd.h>
int main()
{
    int pipefds[2];
    int returnstatus;
    int pid;
    char writemessages[2][20]={"Hi", "Hello"};
    char readmessage[20];
    returnstatus = pipe(pipefds);
    if (returnstatus == -1) {
        printf("Unable to create pipe\n");
        return 1;
    }
    pid = fork();

    // Child process
    if (pid == 0)
    {
        read(pipefds[0], readmessage, sizeof(readmessage));
        printf("Child Process - Reading from pipe – Message 1 is %s\n", readmessage);
        read(pipefds[0], readmessage, sizeof(readmessage));
        printf("Child Process - Reading from pipe – Message 2 is %s\n", readmessage);
    }
    else
    { //Parent process
        printf("Parent Process - Writing to pipe - Message 1 is %s\n", writemessages[0]);
        write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
        printf("Parent Process - Writing to pipe - Message 2 is %s\n", writemessages[1]);
        write(pipefds[1], writemessages[1], sizeof(writemessages[1]));
    }
    return 0;
}
```

Output:

Compilation

gcc pipewithprocesses.c -o pipewithprocesses

Execution

Parent Process - Writing to pipe - Message 1 is Hi

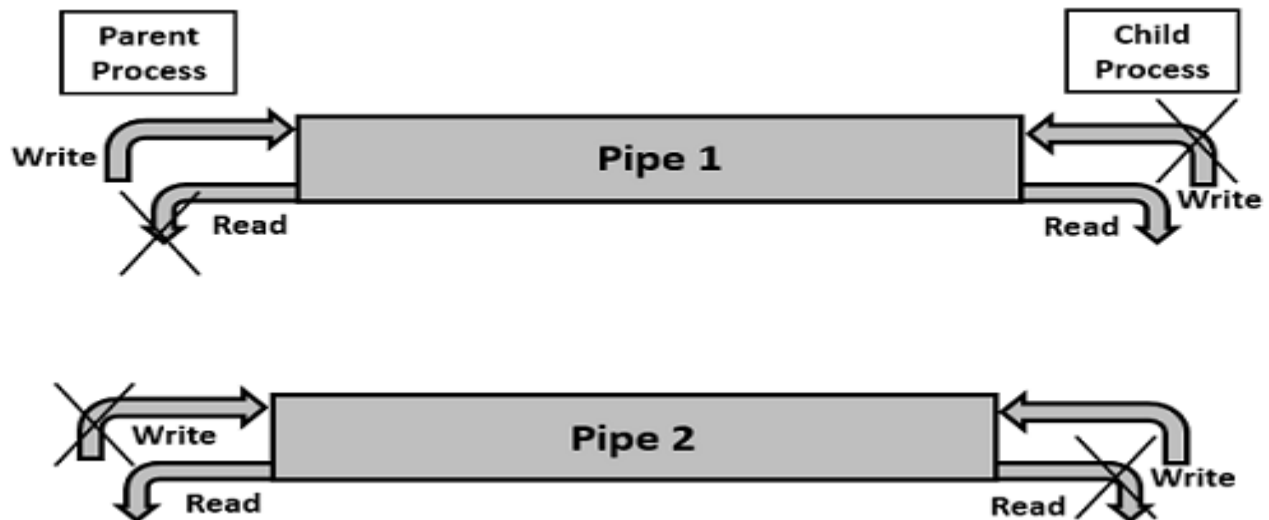
Parent Process - Writing to pipe - Message 2 is Hello

Child Process - Reading from pipe – Message 1 is Hi

Child Process - Reading from pipe – Message 2 is Hello

Cycle-9:

(a) Write a C program for achieving two-way communication using pipes.



Algorithm

- Step 1** – Create pipe1 for the parent process to write and the child process to read.
- Step 2** – Create pipe2 for the child process to write and the parent process to read.
- Step 3** – Close the unwanted ends of the pipe from the parent and child side.
- Step 4** – Parent process to write a message and child process to read and display on the screen.
- Step 5** – Child process to write a message and parent process to read and display on the screen.

Source Code:

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

int main() {
    int pipefds1[2], pipefds2[2];
    int returnstatus1, returnstatus2;
    int pid;
    char pipe1writemessage[20] = "Hi";
    char pipe2writemessage[20] = "Hello";
    char readmessage[20];
    returnstatus1 = pipe(pipefds1);

    if (returnstatus1 == -1) {
        printf("Unable to create pipe 1 \n");
        return 1;
    }
    returnstatus2 = pipe(pipefds2);

    if (returnstatus2 == -1) {
        printf("Unable to create pipe 2 \n");
        return 1;
    }
    pid = fork();

    if (pid != 0) // Parent process {
        close(pipefds1[0]); // Close the unwanted pipe1 read side
        close(pipefds2[1]); // Close the unwanted pipe2 write side
```

```

printf("In Parent: Writing to pipe 1 – Message is %s\n", pipe1writemessage);
write(pipefds1[1], pipe1writemessage, sizeof(pipe1writemessage));
read(pipefds2[0], readmessage, sizeof(readmessage));
printf("In Parent: Reading from pipe 2 – Message is %s\n", readmessage);
} else { //child process
close(pipefds1[1]); // Close the unwanted pipe1 write side
close(pipefds2[0]); // Close the unwanted pipe2 read side
read(pipefds1[0], readmessage, sizeof(readmessage));
printf("In Child: Reading from pipe 1 – Message is %s\n", readmessage);
printf("In Child: Writing to pipe 2 – Message is %s\n", pipe2writemessage);
write(pipefds2[1], pipe2writemessage, sizeof(pipe2writemessage));
}
return 0;
}

```

Output:

Compilation

gcc twowayspipe.c -o twowayspipe

Execution

```

In Parent: Writing to pipe 1 – Message is Hi
In Child: Reading from pipe 1 – Message is Hi
In Child: Writing to pipe 2 – Message is Hello
In Parent: Reading from pipe 2 – Message is Hello

```

Cycle-10

1. Demonstrate with an example the usage of two types of links

A link in UNIX is a pointer to a file. Like pointers in any programming languages, links in UNIX are pointers pointing to a file or a directory. Creating links is a kind of shortcuts to access a file. Links allow more than one file name to refer to the same file, elsewhere.

There are **two types** of links:

1. Soft Link or Symbolic links
2. Hard Links

These links behave differently when the source of the link (what is being linked to) is moved or removed. Symbolic links are not updated (they merely contain a string which is the pathname of its target); hard links always refer to the source, even if moved or removed.

For example, if we have a file a.txt. If we create a hard link to the file and then delete the file, we can still access the file using hard link. But if we create a soft link of the file and then delete the file, we can't access the file through soft link and soft link becomes dangling. Basically hard link increases reference count of a location while soft links work as a shortcut (like in Windows)

1. Hard Links

Each hard linked file is assigned the same Inode value as the original, therefore they reference the same physical file location. Hard links more flexible and remain linked even if the original or linked files are moved throughout the file system, although hard links are unable to cross different file systems.

- ❖ `ls -l` command shows all the links with the link column shows number of links.
- ❖ Links have actual file contents
- ❖ Removing any link, just reduces the link count, but doesn't affect other links.
- ❖ We cannot create a hard link for a directory to avoid recursive loops.
- ❖ If original file is removed then the link will still show the content of the file.
- ❖ Command to create a hard link is:

```
$ ln [original filename] [link name]
```

2. Soft Links

A soft link is similar to the file shortcut feature which is used in Windows Operating systems. Each soft linked file contains a separate Inode value that points to the original file. As similar to hard links, any changes to the data in either file is reflected in the other. Soft links can be linked across different file systems, although if the original file is deleted or moved, the soft linked file will not work correctly (called hanging link).

- ❖ `ls -l` command shows all links with first column value `l?` and the link points to original file.
- ❖ Soft Link contains the path for original file and not the contents.
- ❖ Removing soft link doesn't affect anything but removing original file, the link becomes "dangling" link which points to nonexistent file.
- ❖ A soft link can link to a directory.
- ❖ Link across filesystems: If you want to link files across the filesystems, you can only use symlinks/soft links.
- ❖ Command to create a Soft link is:

```
$ ln -s [original filename] [link name]
```

A **symbolic** or **soft link** is an actual link to the original file, whereas a **hard link** is a mirror copy of the original file. If you delete the original file, the soft link has no value, because it points to a

non-existent file. But in the case of hard link, it is entirely opposite. If you delete the original file, the hard link can still has the data of the original file. Because hard link acts as a mirror copy of the original file.

In a nutshell, a soft link

- can cross the file system,
- allows you to link between directories,
- has different inodes number and file permissions than original file,
- permissions will not be updated,
- has only the path of the original file, not the contents.

A hard Link

- can't cross the file system boundaries,
- can't link directories,
- has the same inodes number and permissions of original file,
- permissions will be updated if we change the permissions of source file,
- has the actual contents of original file, so that you still can view the contents, even if the original file moved or removed.

Still don't get it? Well, allow me to show you some practical examples.

Creating Soft, or Symbolic Link

Let us create an empty directory called **“test”**.

```
$ mkdir test
```

Change to the “test” directory:

```
$ cd test
```

Now, create a new file called **source.file** with some data as shown below.

```
$ echo "Welcome to OSTechNix" >source.file
```

Let us view the data of the source.file.

```
$ cat source.file
Welcome to OSTechNix
```

Well, the source.file has been created.

Now, create the a symbolic or soft link to the source.file.

To do so, run:

```
$ ln -s source.file softlink.file
```

Let us view the data of softlink.file.

```
$ cat softlink.file
```

```
sk@sk: ~/test
File Edit View Search Terminal Help
[sk@sk]: ~/test>$ cat softlink.file
Welcome to OSTechNix
[sk@sk]: ~/test>$
```

As you see in the above output, softlink.file displays the same data as source.file.
Let us check the inodes and permissions of softlink.file and source.file.

```
$ ls -lia
```

Sample output:

```
total 12
15745326 drwxr-xr-x 2 sk users 4096 Dec 13 14:55 .
15728642 drwx----- 49 sk users 4096 Dec 13 14:50 ..
15746561 lrwxrwxrwx 1 sk users 11 Dec 13 14:55 softlink.file -> source.file
15746185 -rw-r--r-- 1 sk users 21 Dec 13 14:53 source.file
```

```
sk@sk: ~/test
File Edit View Search Terminal Help
[sk@sk]: ~/test>$ ls -lia
total 12
15745326 drwxr-xr-x 2 sk users 4096 Dec 13 14:55 .
15728642 drwx----- 49 sk users 4096 Dec 13 14:50 ..
15746561 lrwxrwxrwx 1 sk users 11 Dec 13 14:55 softlink.file -> source.file
15746185 -rw-r--r-- 1 sk users 21 Dec 13 14:53 source.file
[sk@sk]: ~/test>$
```

inodes number **File permission**

As we see in the above screenshot, even though the softlink.file has same contents as source.file, the **inodes number** (15746561 vs 15746185) and **file permissions** (lrwxrwxrwx vs -rw-r--r-) are different. Hence, it is proved that soft link doesn't share the same inodes number and permissions of original file.

Now, remove the original file (i.e source.file) and see what happens.

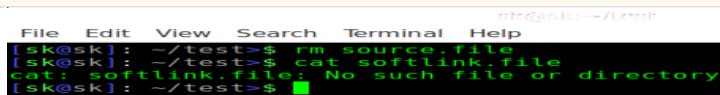
```
$ rm source.file
```

Check output of softlink.file using command:

```
$ cat softlink.file
```

Sample output:

```
cat: softlink.file: No such file or directory
```



```
File Edit View Search Terminal Help
[sk@sk]: ~/test>$ rm source.file
[sk@sk]: ~/test>$ cat softlink.file
cat: softlink.file: No such file or directory
[sk@sk]: ~/test>$
```

As you see above, there is no such file or directory called softlink.file after we removed the original file (i.e source.file). So, now we understand that soft link is just a link that points to the original file. The softlink is like a shortcut to a file. If you remove the file, the shortcut is useless. As you already know, if you remove the soft link, the original file will still present.

Creating Hard Link

Create a file called **source.file** with some contents as shown below.

```
$ echo "Welcome to OSTechNix" >source.file
```

Let us verify the contents of the file.

```
$ cat source.file
```

```
Welcome to OSTechNix
```

source.file has been created now.

Now, let us create the hard link to the source.file as shown below.

```
$ ln source.file hardlink.file
```



```
File Edit View Search Terminal Help
[sk@sk]: ~/test>$ ln source.file hardlink.file
[sk@sk]: ~/test>$
```

Check the contents of hardlink.file.

```
$ cat hardlink.file
```

```
Welcome to OSTechNix
```

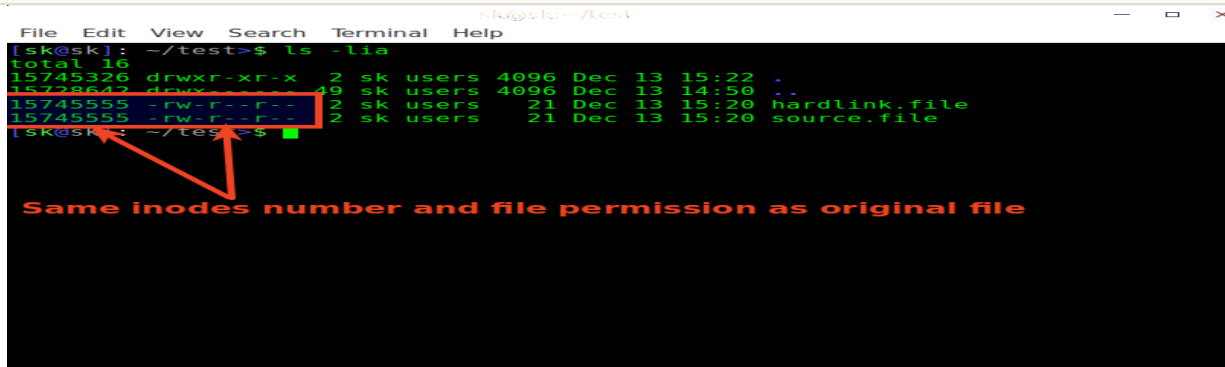
You see the hardlink.file displays the same data as source.file.

Let us check the inodes and permissions of softlink.file and source.file.

```
$ ls -lia
```

Sample output:

```
total 16
15745326 drwxr-xr-x 2 sk users 4096 Dec 13 15:22 .
15728642 drwx----- 49 sk users 4096 Dec 13 14:50 ..
15745555 -rw-r--r-- 2 sk users 21 Dec 13 15:20 hardlink.file
15745555 -rw-r--r-- 2 sk users 21 Dec 13 15:20 source.file
```



```
File Edit View Search Terminal Help
[sk@sk]: ~/test>$ ls -lia
total 16
15745326 drwxr-xr-x 2 sk users 4096 Dec 13 15:22 .
15728642 drwx----- 49 sk users 4096 Dec 13 14:50 ..
15745555 -rw-r--r-- 2 sk users 21 Dec 13 15:20 hardlink.file
15745555 -rw-r--r-- 2 sk users 21 Dec 13 15:20 source.file
[sk@sk]: ~/test>$
```

Same inodes number and file permission as original file

Now, we see that both hardlink.file and source.file have the same the **inodes number (15745555)** and **file permissions (-rw-r--r-)**. Hence, it is proved that hard link file shares the same inodes number and permissions of original file.

Note: If we change the permissions on source.file, the same permission will be applied to the hardlink.file as well.

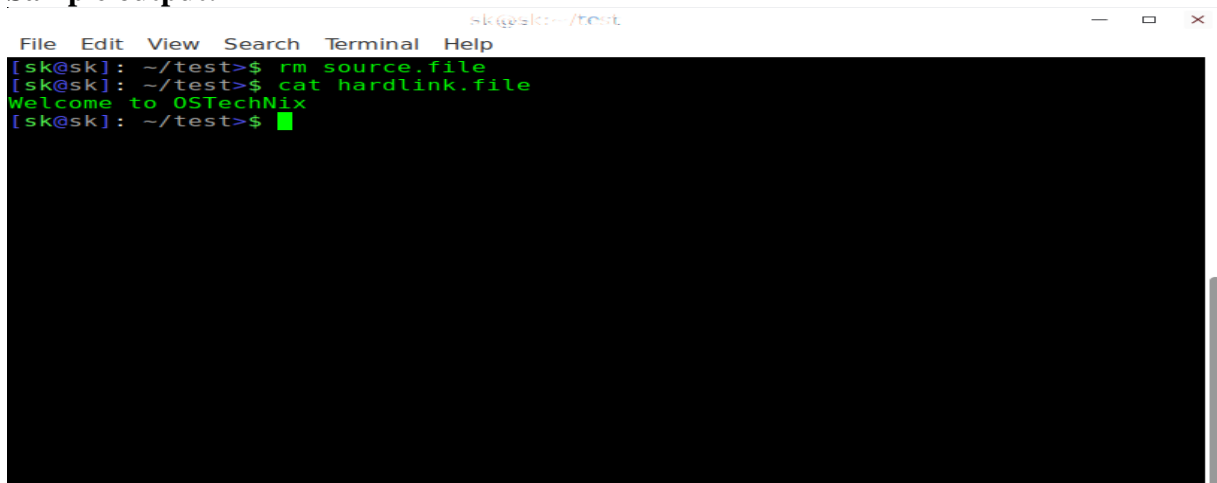
Now, remove the original file (i.e source.file) and see what happens.

```
$ rm source.file
```

Check contents of hardlink.file using command:

```
$ cat hardlink.file
```

Sample output:



```
File Edit View Search Terminal Help
[sk@sk]: ~/test>$ rm source.file
[sk@sk]: ~/test>$ cat hardlink.file
Welcome to OSTechNix
[sk@sk]: ~/test>$
```

As you see above, even if I deleted the source file, I can view contents of the hardlink.file. Hence, it is proved that Hard link shares the same inodes number, the permissions and data of the original file.