Operating Systems Lab - L3+L4
Lab Assessment 1

Name: BHAVIKK D PATEL
Register No. : 24MCS0047

1. Study the fundamental Unix/Linux commands.

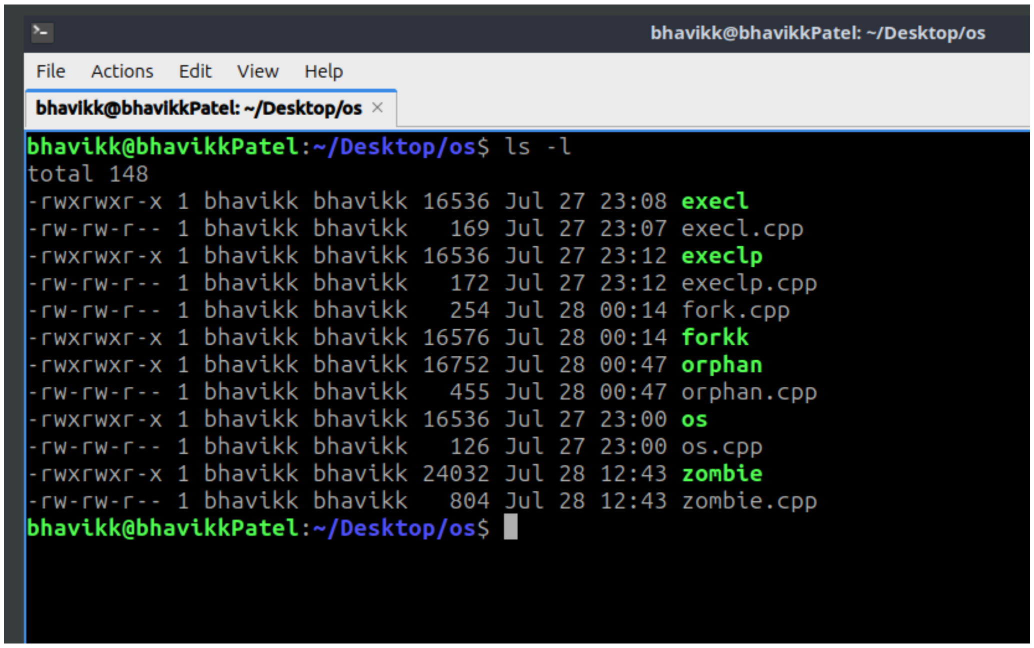
#1

Command Name	:	ls
Description	:	lists all the files which are in the current directory
Syntax	:	ls -[OPTIONS]
Example	:	ls -l

This is a list command which lists all the files which are in the current directory.
From the output we can see the permissions which are available for each file.
"rwx" is:

- r - read
- w - write
- x - executable

Files which are marked with 'x' are executable files.


bhavikk@bhavikkPatel: ~/Desktop/os
File Actions Edit View Help
bhavikk@bhavikkPatel: ~/Desktop/os

```
bhavikk@bhavikkPatel:~/Desktop/os$ ls -l
total 148
-rwxrwxr-x 1 bhavikk bhavikk 16536 Jul 27 23:08 execl
-rw-rw-r-- 1 bhavikk bhavikk   169 Jul 27 23:07 execl.cpp
-rwxrwxr-x 1 bhavikk bhavikk 16536 Jul 27 23:12 execlp
-rw-rw-r-- 1 bhavikk bhavikk   172 Jul 27 23:12 execlp.cpp
-rw-rw-r-- 1 bhavikk bhavikk   254 Jul 28 00:14 fork.cpp
-rwxrwxr-x 1 bhavikk bhavikk 16576 Jul 28 00:14 forkk
-rwxrwxr-x 1 bhavikk bhavikk 16752 Jul 28 00:47 orphan
-rw-rw-r-- 1 bhavikk bhavikk   455 Jul 28 00:47 orphan.cpp
-rwxrwxr-x 1 bhavikk bhavikk 16536 Jul 27 23:00 os
-rw-rw-r-- 1 bhavikk bhavikk   126 Jul 27 23:00 os.cpp
-rwxrwxr-x 1 bhavikk bhavikk 24032 Jul 28 12:43 zombie
-rw-rw-r-- 1 bhavikk bhavikk   804 Jul 28 12:43 zombie.cpp
bhavikk@bhavikkPatel:~/Desktop/os$
```

#2

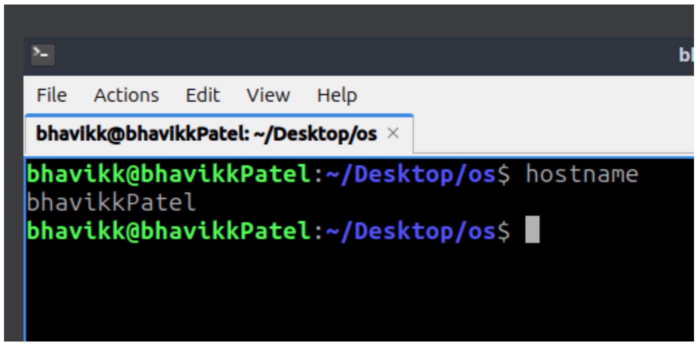
Command Name : hostname

Description : shows the name of the current host system

Syntax : hostname

Example : hostname

It is useful when in person is working with multiple Cloud PC connected via terminal.


File Actions Edit View Help
bhavikk@bhavikkPatel: ~/Desktop/os

```
bhavikk@bhavikkPatel:~/Desktop/os$ hostname
bhavikkPatel
bhavikk@bhavikkPatel:~/Desktop/os$
```

#3

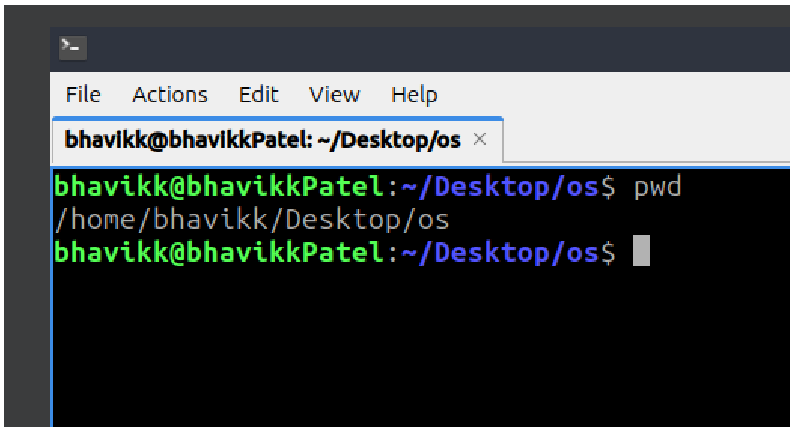
Command Name : pwd

Description : print the full path name of your current directory

Syntax : pwd

Example : pwd

Full path starts from the root directory.


File Actions Edit View Help
bhavikk@bhavikkPatel: ~/Desktop/os

```
bhavikk@bhavikkPatel:~/Desktop/os$ pwd
/home/bhavikk/Desktop/os
bhavikk@bhavikkPatel:~/Desktop/os$
```

#4

Command Name : ps

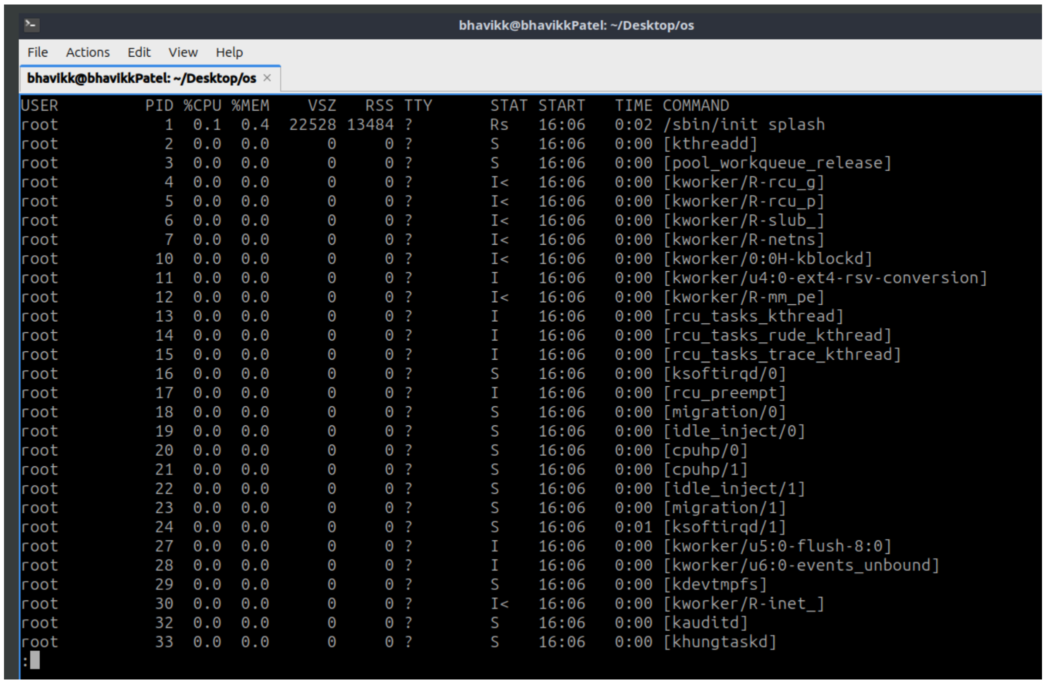
Description : information about currently running processes in the system.

Syntax : ps

Example : ps -aux

Each process has a PID and PPID

- PID: process ID
- PPID: parent process ID

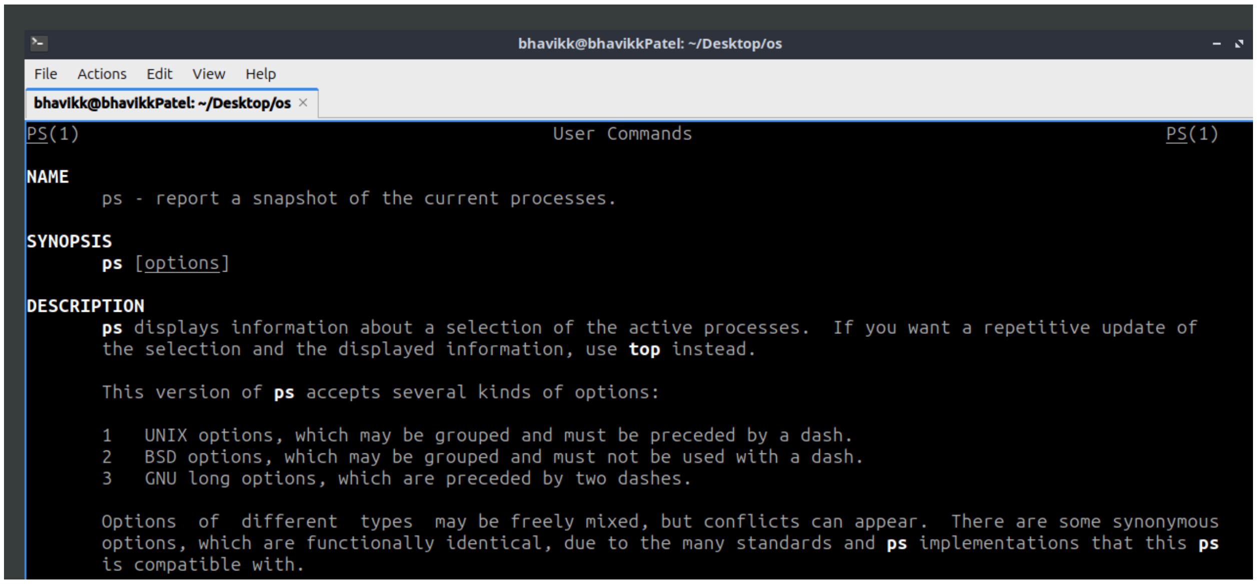

bhavikk@bhavikkPatel: ~/Desktop/os
File Actions Edit View Help
bhavikk@bhavikkPatel: ~/Desktop/os

```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.1  0.4  22528 13484 ?        Rs   16:06   0:02 /sbin/init splash
root           2  0.0  0.0      0     0 ?        S    16:06   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        S    16:06   0:00 [pool_workqueue_release]
root           4  0.0  0.0      0     0 ?        I<   16:06   0:00 [kworker/R-rcu_g]
root           5  0.0  0.0      0     0 ?        I<   16:06   0:00 [kworker/R-rcu_p]
root           6  0.0  0.0      0     0 ?        I<   16:06   0:00 [kworker/R-slub_]
root           7  0.0  0.0      0     0 ?        I<   16:06   0:00 [kworker/R-netns]
root          10  0.0  0.0      0     0 ?        I<   16:06   0:00 [kworker/0:0H-kblockd]
root          11  0.0  0.0      0     0 ?        I    16:06   0:00 [kworker/u4:0-ext4-rsv-conversion]
root          12  0.0  0.0      0     0 ?        I<   16:06   0:00 [kworker/R-mm_pe]
root          13  0.0  0.0      0     0 ?        I    16:06   0:00 [rcu_tasks_kthread]
root          14  0.0  0.0      0     0 ?        I    16:06   0:00 [rcu_tasks_rude_kthread]
root          15  0.0  0.0      0     0 ?        I    16:06   0:00 [rcu_tasks_trace_kthread]
root          16  0.0  0.0      0     0 ?        S    16:06   0:00 [ksoftirqd/0]
root          17  0.0  0.0      0     0 ?        I    16:06   0:00 [rcu_preempt]
root          18  0.0  0.0      0     0 ?        S    16:06   0:00 [migration/0]
root          19  0.0  0.0      0     0 ?        S    16:06   0:00 [idle_inject/0]
root          20  0.0  0.0      0     0 ?        S    16:06   0:00 [cpuhp/0]
root          21  0.0  0.0      0     0 ?        S    16:06   0:00 [cpuhp/1]
root          22  0.0  0.0      0     0 ?        S    16:06   0:00 [idle_inject/1]
root          23  0.0  0.0      0     0 ?        S    16:06   0:00 [migration/1]
root          24  0.0  0.0      0     0 ?        S    16:06   0:01 [ksoftirqd/1]
root          27  0.0  0.0      0     0 ?        I    16:06   0:00 [kworker/u5:0-flush-8:0]
root          28  0.0  0.0      0     0 ?        I    16:06   0:00 [kworker/u6:0-events_unbound]
root          29  0.0  0.0      0     0 ?        S    16:06   0:00 [kdevtmpfs]
root          30  0.0  0.0      0     0 ?        I<   16:06   0:00 [kworker/R-inet_]
root          32  0.0  0.0      0     0 ?        S    16:06   0:00 [kauditd]
root          33  0.0  0.0      0     0 ?        S    16:06   0:00 [khungtaskd]
:
```

#5

Command Name	**:**	man
Description	**:**	display user manual of any command that run on terminal
Syntax	**:**	man <command name>
Example	**:**	man ps

Provides detailed description with the synopsis to refer. This command is especially useful when working with system calls.


bhavikk@bhavikkPatel: ~/Desktop/os
File Actions Edit View Help
bhavikk@bhavikkPatel: ~/Desktop/os

```
PS(1)                                  User Commands                                  PS(1)

NAME
       ps - report a snapshot of the current processes.

SYNOPSIS
       ps [options]

DESCRIPTION
       ps displays information about a selection of the active processes.  If you want a repetitive update of
       the selection and the displayed information, use top instead.

       This version of ps accepts several kinds of options:

       1   UNIX options, which may be grouped and must be preceded by a dash.
       2   BSD options, which may be grouped and must not be used with a dash.
       3   GNU long options, which are preceded by two dashes.

       Options  of  different  types  may be freely mixed, but conflicts can appear.  There are some synonymous
       options, which are functionally identical, due to the many standards and ps implementations that this ps
       is compatible with.
```

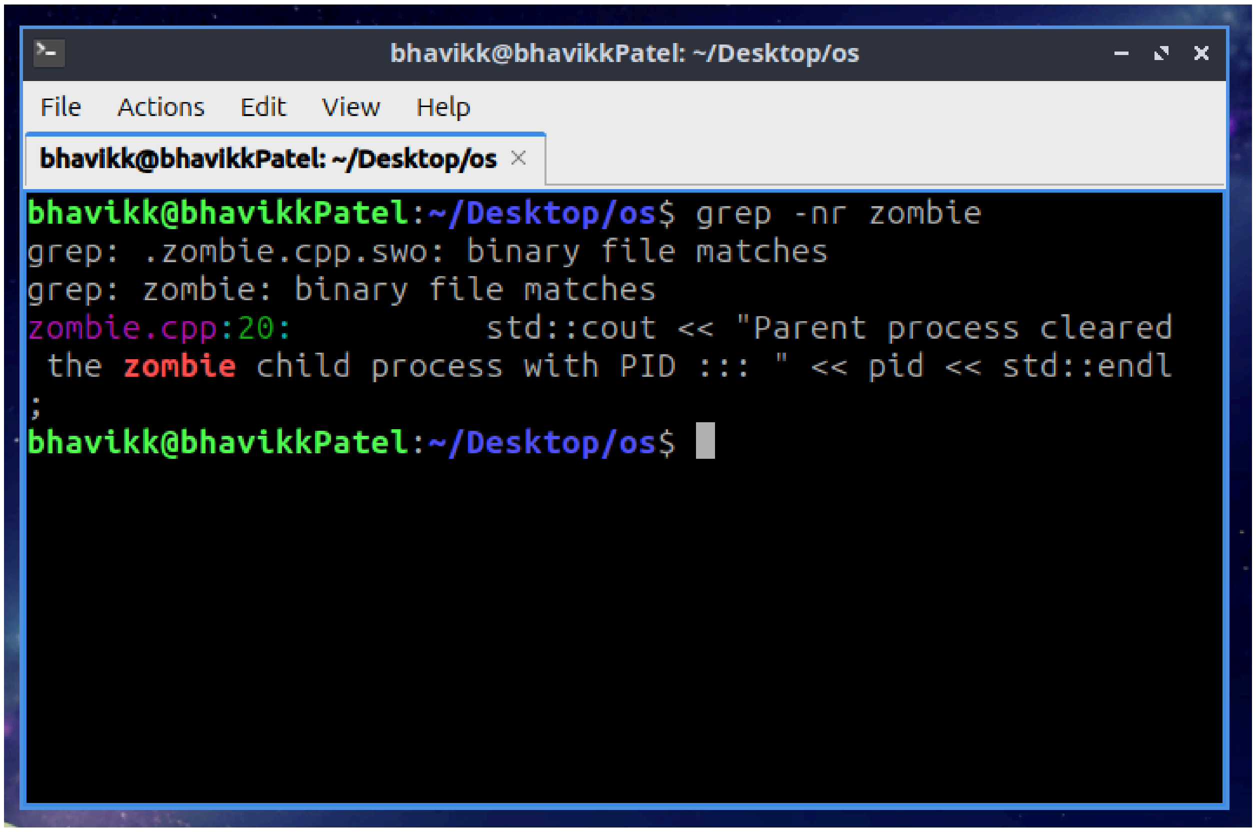
#6

Command Name : grep

Description : globally search for regular expression and print matching line

Syntax : grep [options] pattern [files]

Example : grep -nr zombie


bhavikk@bhavikkPatel: ~/Desktop/os
File
Actions
Edit
View
Help
bhavikk@bhavikkPatel: ~/Desktop/os

```
bhavikk@bhavikkPatel:~/Desktop/os$ grep -nr zombie
grep: .zombie.cpp.swo: binary file matches
grep: zombie: binary file matches
zombie.cpp:20:            std::cout << "Parent process cleared
 the zombie child process with PID ::: " << pid << std::endl
;
bhavikk@bhavikkPatel:~/Desktop/os$ 
```

#7

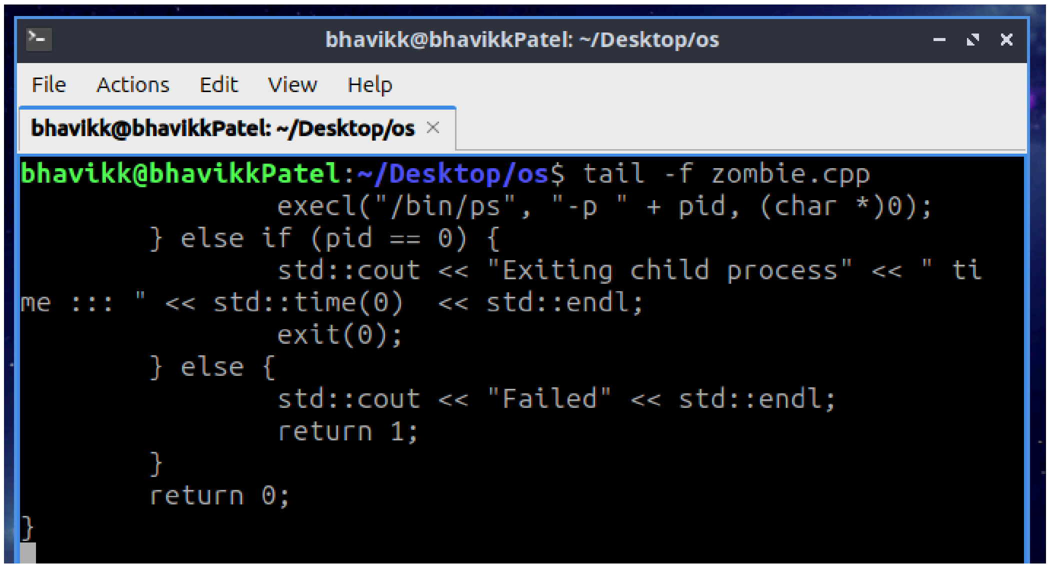
Command Name : tail

Description : displays a specified number of trailing lines from the specified file

Syntax : tail [options] path/filename

Example : tail -f zombie.cpp

Example: Tail with -f flag updates the output in the terminal automatically when the file gets updated. Command is very useful for developers especially when working in dev setup and referring to the log file which continuously gets updated by the application.


bhavikk@bhavikkPatel: ~/Desktop/os
File Actions Edit View Help
bhavikk@bhavikkPatel: ~/Desktop/os

```
bhavikk@bhavikkPatel:~/Desktop/os$ tail -f zombie.cpp
                execl("/bin/ps", "-p " + pid, (char *)0);
        } else if (pid == 0) {
                std::cout << "Exiting child process" << " ti
me ::: " << std::time(0)  << std::endl;
                exit(0);
        } else {
                std::cout << "Failed" << std::endl;
                return 1;
        }
        return 0;
}
```

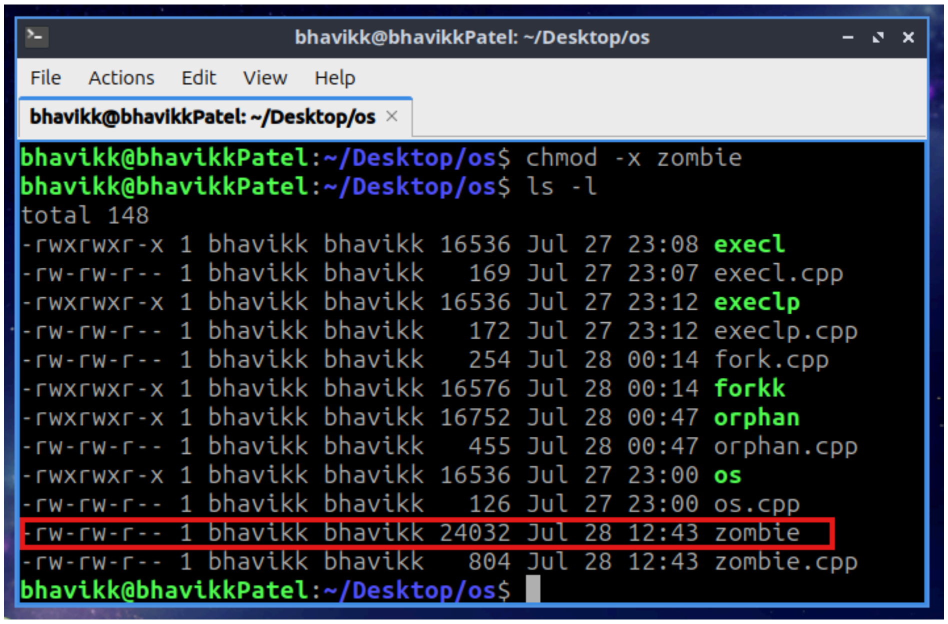
#8

Command Name : chmod

Description : command is used to change the permissions of a file or directory

Syntax : chmod [options] filename/directory

Example : chmod -x zombie


bhavikk@bhavikkPatel: ~/Desktop/os
File Actions Edit View Help
bhavikk@bhavikkPatel: ~/Desktop/os

```
bhavikk@bhavikkPatel:~/Desktop/os$ chmod -x zombie
bhavikk@bhavikkPatel:~/Desktop/os$ ls -l
total 148
-rwxrwxr-x 1 bhavikk bhavikk 16536 Jul 27 23:08 execl
-rw-rw-r-- 1 bhavikk bhavikk   169 Jul 27 23:07 execl.cpp
-rwxrwxr-x 1 bhavikk bhavikk 16536 Jul 27 23:12 execlp
-rw-rw-r-- 1 bhavikk bhavikk   172 Jul 27 23:12 execlp.cpp
-rw-rw-r-- 1 bhavikk bhavikk   254 Jul 28 00:14 fork.cpp
-rwxrwxr-x 1 bhavikk bhavikk 16576 Jul 28 00:14 forkk
-rwxrwxr-x 1 bhavikk bhavikk 16752 Jul 28 00:47 orphan
-rw-rw-r-- 1 bhavikk bhavikk   455 Jul 28 00:47 orphan.cpp
-rwxrwxr-x 1 bhavikk bhavikk 16536 Jul 27 23:00 os
-rw-rw-r-- 1 bhavikk bhavikk   126 Jul 27 23:00 os.cpp
-rw-rw-r-- 1 bhavikk bhavikk 24032 Jul 28 12:43 zombie
-rw-rw-r-- 1 bhavikk bhavikk   804 Jul 28 12:43 zombie.cpp
bhavikk@bhavikkPatel:~/Desktop/os$
```

Example: From the above image, we can see that after executing the command "chmod -x zombie", the executable permission is taken away from the file.

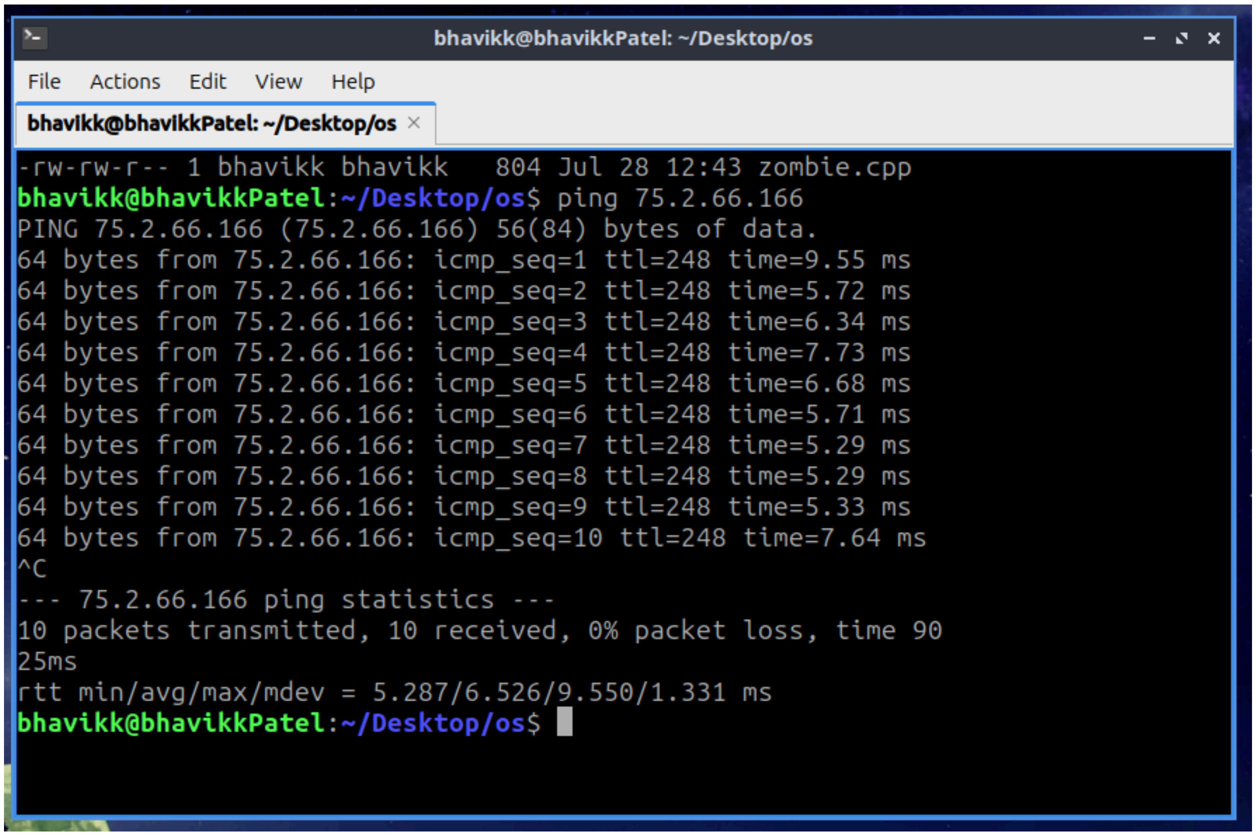
#9

Command Name : ping

Description : check the network connectivity between the host and server/host

Syntax : ping [options] host_or_IP_address

Example : ping 75.2.66.166


bhavikk@bhavikkPatel: ~/Desktop/os
File Actions Edit View Help
bhavikk@bhavikkPatel: ~/Desktop/os

```
-rw-rw-r-- 1 bhavikk bhavikk   804 Jul 28 12:43 zombie.cpp
bhavikk@bhavikkPatel:~/Desktop/os$ ping 75.2.66.166
PING 75.2.66.166 (75.2.66.166) 56(84) bytes of data.
64 bytes from 75.2.66.166: icmp_seq=1 ttl=248 time=9.55 ms
64 bytes from 75.2.66.166: icmp_seq=2 ttl=248 time=5.72 ms
64 bytes from 75.2.66.166: icmp_seq=3 ttl=248 time=6.34 ms
64 bytes from 75.2.66.166: icmp_seq=4 ttl=248 time=7.73 ms
64 bytes from 75.2.66.166: icmp_seq=5 ttl=248 time=6.68 ms
64 bytes from 75.2.66.166: icmp_seq=6 ttl=248 time=5.71 ms
64 bytes from 75.2.66.166: icmp_seq=7 ttl=248 time=5.29 ms
64 bytes from 75.2.66.166: icmp_seq=8 ttl=248 time=5.29 ms
64 bytes from 75.2.66.166: icmp_seq=9 ttl=248 time=5.33 ms
64 bytes from 75.2.66.166: icmp_seq=10 ttl=248 time=7.64 ms
^C
--- 75.2.66.166 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 90
25ms
rtt min/avg/max/mdev = 5.287/6.526/9.550/1.331 ms
bhavikk@bhavikkPatel:~/Desktop/os$
```

Example: Following is riot's Mumbai server's IP. We can ping and check the response time and can also get possible packet loss data.

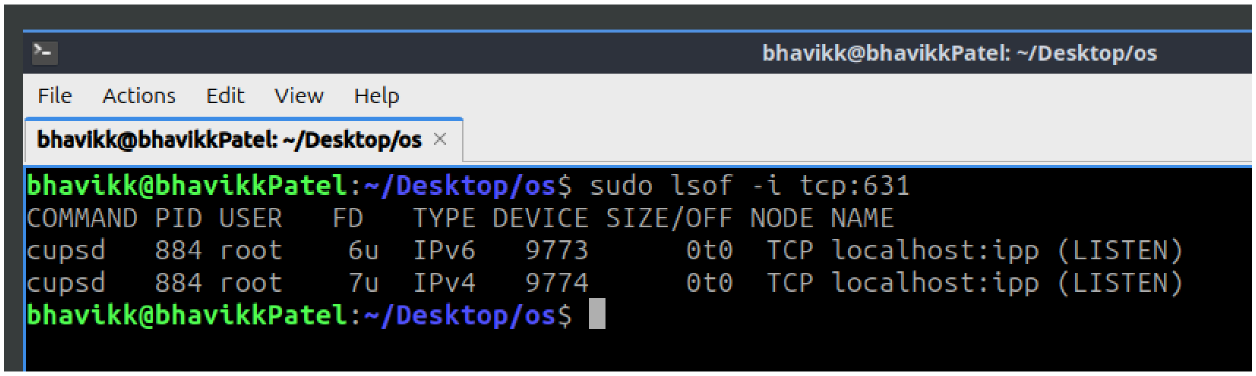
#10

Command Name : lsof

Description : "list open files" and displays all the files that your system has currently opened.

Syntax : lsof [options] [names]

Example : lsof -i tcp:631


bhavikk@bhavikkPatel: ~/Desktop/os
File Actions Edit View Help
bhavikk@bhavikkPatel: ~/Desktop/os

```
bhavikk@bhavikkPatel:~/Desktop/os$ sudo lsof -i tcp:631
COMMAND PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
cupsd   884 root    6u  IPv6   9773      0t0  TCP localhost:ipp (LISTEN)
cupsd   884 root    7u  IPv4   9774      0t0  TCP localhost:ipp (LISTEN)
bhavikk@bhavikkPatel:~/Desktop/os$
```

Example: This command with -i tcp:<port> options is very useful for developers when they are working with server setups running in a local machine and requires to check if the port is being used or not, if yes, then information about the process using the port is specified.

2. **Study of differences between system( ) and execl( ) /execlp( ) calls. Give examples.**

System() function:

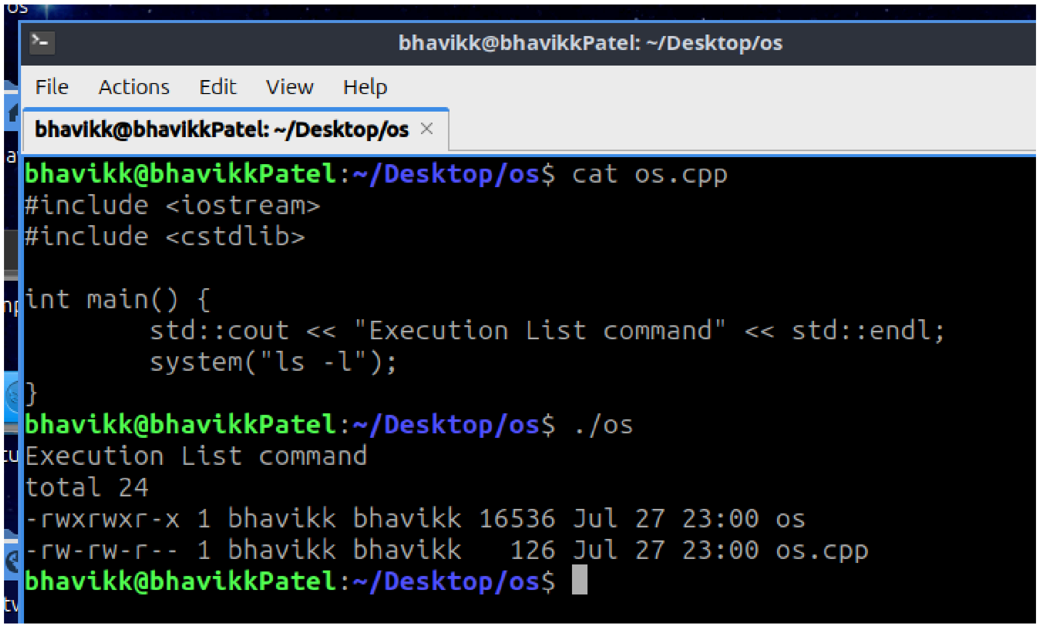
The system() function is used to execute a shell command from within a C/C++ program. It uses the system's default command interpreter to run the specified command.

- The system() function invokes the command processor to execute a command.
- It forks a new process, which then invokes the command processor like /bin/sh on unix based systems to run the specified command.
- The command processor interprets the command, executes it and returns the output to the called program.

This is mainly used for executing simple shell commands which do not have security concerns.

Code:

```
#include <iostream>
#include <cstdlib>
// Bhavikk Patel
// 24MCS0047
int main() {
        std::cout << "Execution List command" << std::endl;
        system("ls -l");
}
```



bhavikk@bhavikkPatel: ~/Desktop/os
File Actions Edit View Help
bhavikk@bhavikkPatel: ~/Desktop/os

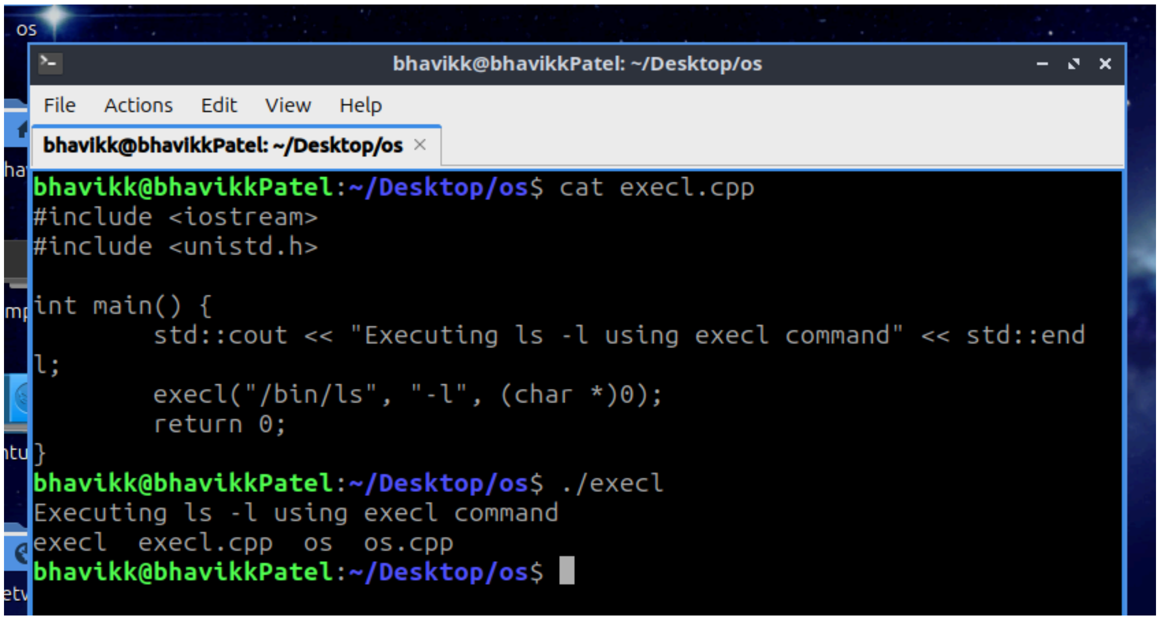
```
bhavikk@bhavikkPatel:~/Desktop/os$ cat os.cpp
#include <iostream>
#include <cstdlib>

int main() {
        std::cout << "Execution List command" << std::endl;
        system("ls -l");
}
bhavikk@bhavikkPatel:~/Desktop/os$ ./os
Execution List command
total 24
-rwxrwxr-x 1 bhavikk bhavikk 16536 Jul 27 23:00 os
-rw-rw-r-- 1 bhavikk bhavikk   126 Jul 27 23:00 os.cpp
bhavikk@bhavikkPatel:~/Desktop/os$
```

execl() function:

The execl() function is used to replace the current process with a new process specified by a file path. It requires the full path to the executable and does not return unless there is an error.

- The execl() function replaces the current process with a new process.
- It does not create a new process but overlays the current process with the new process image specified by the executable file.
- The arguments to execl() are the path to the executable and a list of arguments ending with a NULL pointer.
- Little complex to use when compared to system() function.
- It provides more control over the executed command as does not invoke a shell which results in reducing security risks.

Code:
```
#include <iostream>
#include <unistd.h>
// Bhavikk Patel
// 24MCS0047
int main() {
        std::cout << "Executing ls -l using execl command" << std::endl;
        execl("/bin/ls", "-l", (char *)0);
        return 0;
}
```


bhavikk@bhavikkPatel: ~/Desktop/os
File Actions Edit View Help
bhavikk@bhavikkPatel: ~/Desktop/os

```
bhavikk@bhavikkPatel:~/Desktop/os$ cat execl.cpp
#include <iostream>
#include <unistd.h>

int main() {
        std::cout << "Executing ls -l using execl command" << std::end
l;
        execl("/bin/ls", "-l", (char *)0);
        return 0;
}
bhavikk@bhavikkPatel:~/Desktop/os$ ./execl
Executing ls -l using execl command
execl  execl.cpp  os  os.cpp
bhavikk@bhavikkPatel:~/Desktop/os$
```

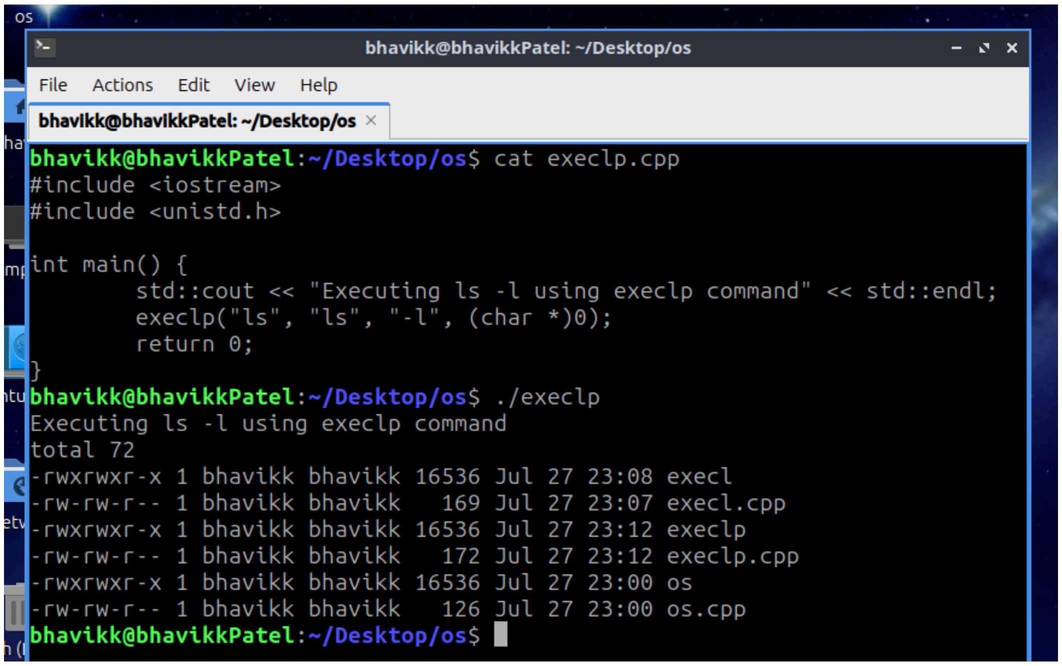
execlp() function:

The execlp() function is similar to execl() but searches for the executable in the directories listed in the PATH environment variable. This makes it more flexible as it doesn't need the full path to the executable.

- The execlp() function replaces the current process with a new process specified by the executable name.
- It searches for the executable in the directories listed in the PATH environment variable.
- The arguments to execlp() are the executable name and a list of arguments ending with a NULL pointer.

Code:

```
#include <iostream>
#include <unistd.h>
// Bhavikk Patel
// 24MCS0047
int main() {
        std::cout << "Executing ls -l using execlp command" << std::endl;
        execlp("ls", "ls", "-l", (char *)0);
        return 0;
}
```


bhavikk@bhavikkPatel: ~/Desktop/os
File Actions Edit View Help
bhavikk@bhavikkPatel: ~/Desktop/os

```
bhavikk@bhavikkPatel:~/Desktop/os$ cat execlp.cpp
#include <iostream>
#include <unistd.h>

int main() {
        std::cout << "Executing ls -l using execlp command" << std::endl;
        execlp("ls", "ls", "-l", (char *)0);
        return 0;
}
bhavikk@bhavikkPatel:~/Desktop/os$ ./execlp
Executing ls -l using execlp command
total 72
-rwxrwxr-x 1 bhavikk bhavikk 16536 Jul 27 23:08 execl
-rw-rw-r-- 1 bhavikk bhavikk   169 Jul 27 23:07 execl.cpp
-rwxrwxr-x 1 bhavikk bhavikk 16536 Jul 27 23:12 execlp
-rw-rw-r-- 1 bhavikk bhavikk   172 Jul 27 23:12 execlp.cpp
-rwxrwxr-x 1 bhavikk bhavikk 16536 Jul 27 23:00 os
-rw-rw-r-- 1 bhavikk bhavikk   126 Jul 27 23:00 os.cpp
bhavikk@bhavikkPatel:~/Desktop/os$
```

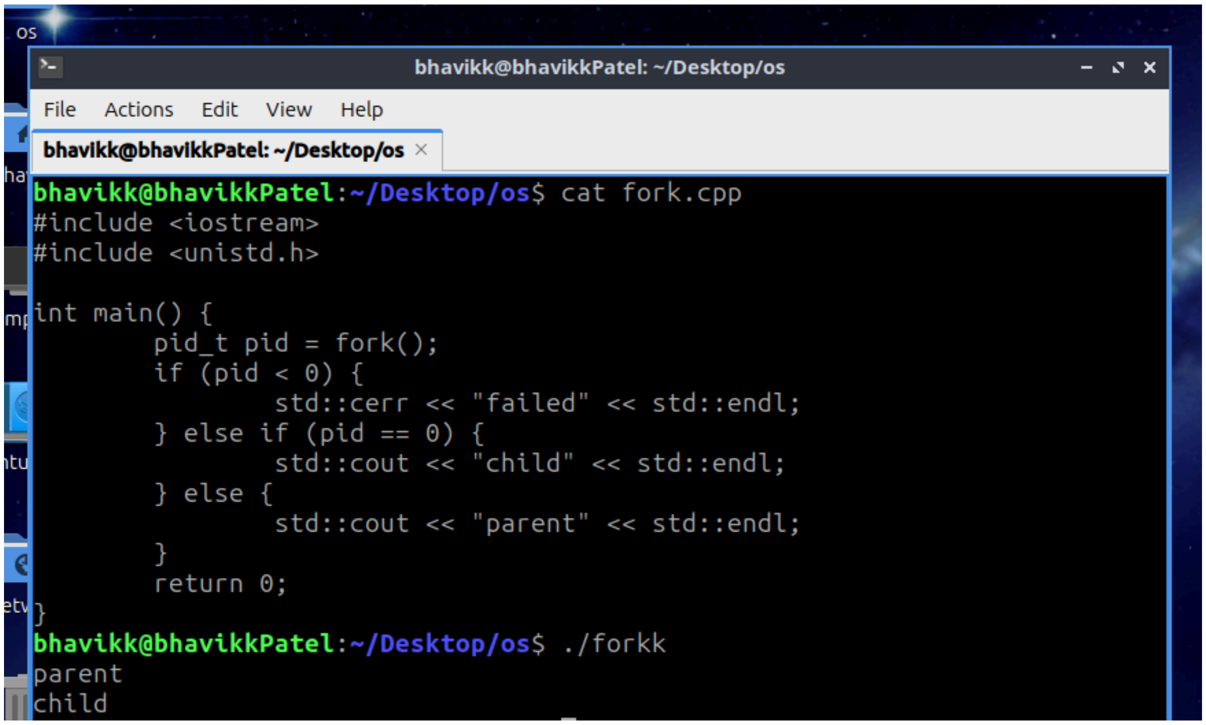
3. Study on fork( ) system call

The fork() system call is a fundamental operation in Unix-like operating systems for creating a new process, often referred to as a child process. It's a powerful tool for creating concurrent execution paths within a program.

- In the parent process, fork() returns the process ID (PID) of the child process.
- In the child process, fork() returns 0.
- If an error occurs, fork() returns -1.
- The child process is an exact copy of the parent process at the time of the fork() call, but it gets its own unique PID and PPID.
- Also, The parent and child processes execute concurrently and independently.

Code:

```
#include <iostream>
#include <unistd.h>
// Bhavikk Patel
// 24MCS0047
int main() {
        pid_t pid = fork();
        if (pid < 0) {
        std::cerr << "failed" << std::endl;
        } else if (pid == 0) {
        std::cout << "child" << std::endl;
        } else {
        std::cout << "parent" << std::endl;
        }
        return 0;
}
```


bhavikk@bhavikkPatel: ~/Desktop/os
File Actions Edit View Help
bhavikk@bhavikkPatel: ~/Desktop/os

```
bhavikk@bhavikkPatel:~/Desktop/os$ cat fork.cpp
#include <iostream>
#include <unistd.h>

int main() {
        pid_t pid = fork();
        if (pid < 0) {
                std::cerr << "failed" << std::endl;
        } else if (pid == 0) {
                std::cout << "child" << std::endl;
        } else {
                std::cout << "parent" << std::endl;
        }
        return 0;
}
bhavikk@bhavikkPatel:~/Desktop/os$ ./forkk
parent
child
```

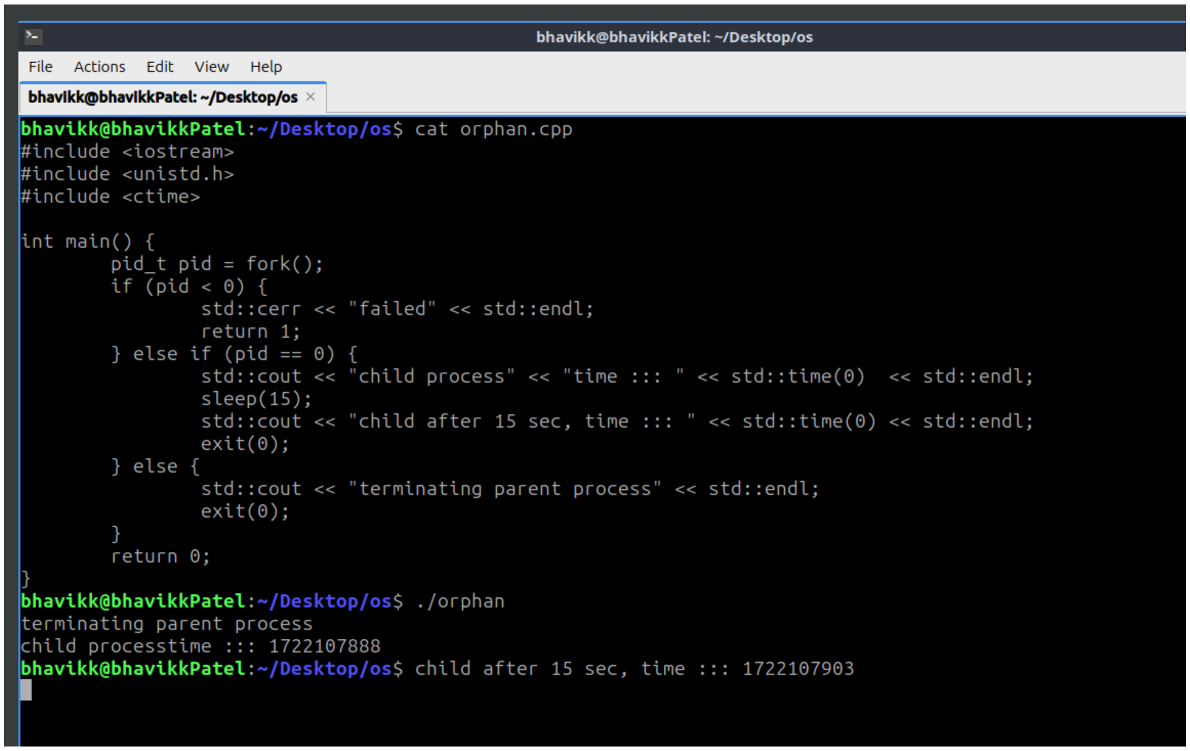
- In the above example, a child process is forked from the parent process.
- It is observed that, child process has the same code as the parent process and it prints the output “child”

4. **How would you conceptualize an Orphan process? How do we locate an orphan process through 'ps' command? Write a C Program to create an Orphan process.**

An orphan process is a process whose parent process has terminated before the child process. When this happens, the operating system typically adopts the orphan process, making the init process (PID 1) its new parent.

Code:

```
#include <iostream>
#include <unistd.h>
#include <ctime>
// Bhavikk Patel
// 24MCS0047
int main() {
        pid_t pid = fork();
        if (pid < 0) {
        std::cerr << "failed" << std::endl;
        return 1;
        } else if (pid == 0) {
        std::cout << "child process" << "time ::: " << std::time(0)  << std::endl;
        sleep(15);
        std::cout << "child after 15 sec, time ::: " << std::time(0) << std::endl;
        exit(0);
        } else {
        std::cout << "terminating parent process" << std::endl;
        exit(0);
        }
        return 0;
}
```


bhavikk@bhavikkPatel: ~/Desktop/os
File Actions Edit View Help
bhavikk@bhavikkPatel: ~/Desktop/os

```
bhavikk@bhavikkPatel:~/Desktop/os$ cat orphan.cpp
#include <iostream>
#include <unistd.h>
#include <ctime>

int main() {
        pid_t pid = fork();
        if (pid < 0) {
                std::cerr << "failed" << std::endl;
                return 1;
        } else if (pid == 0) {
                std::cout << "child process" << "time ::: " << std::time(0)  << std::endl;
                sleep(15);
                std::cout << "child after 15 sec, time ::: " << std::time(0) << std::endl;
                exit(0);
        } else {
                std::cout << "terminating parent process" << std::endl;
                exit(0);
        }
        return 0;
}
bhavikk@bhavikkPatel:~/Desktop/os$ ./orphan
terminating parent process
child processtime ::: 1722107888
bhavikk@bhavikkPatel:~/Desktop/os$ child after 15 sec, time ::: 1722107903
```

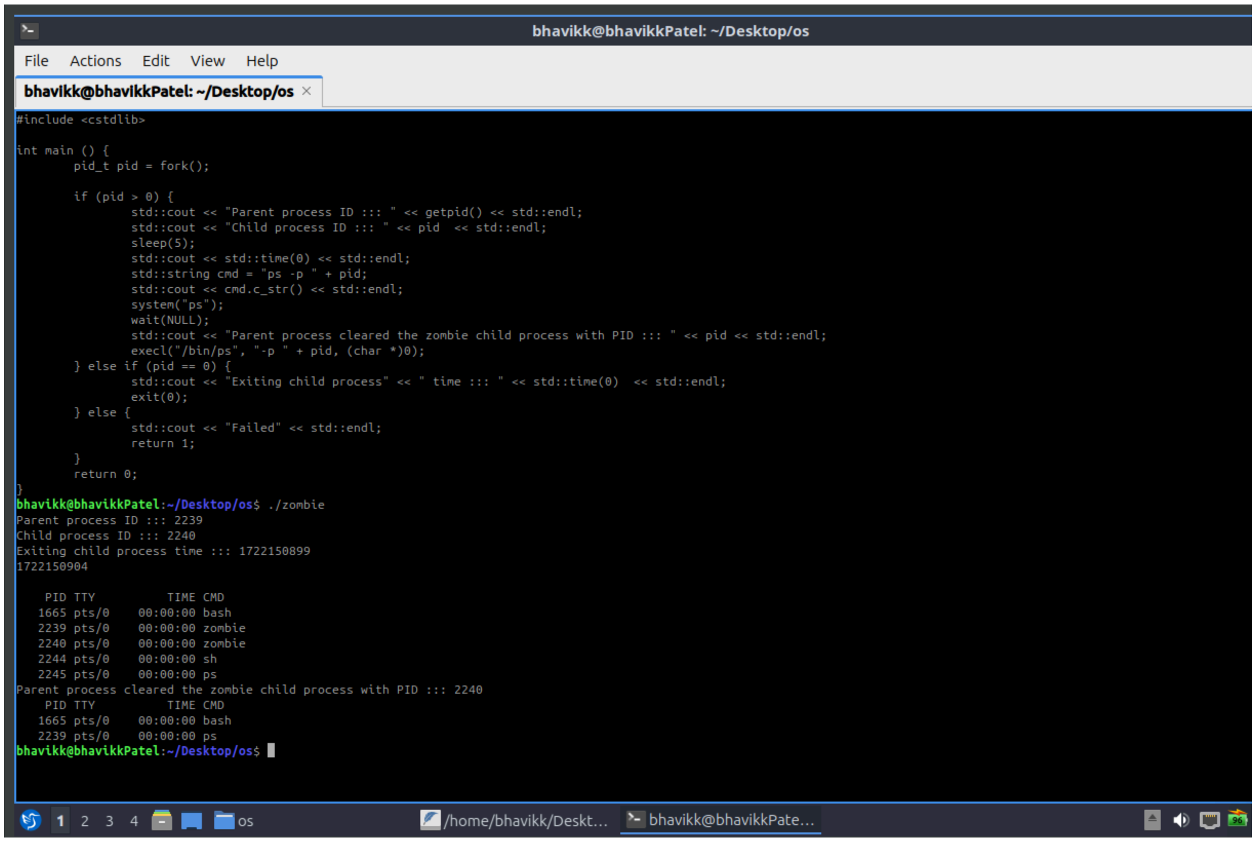
- In the above example, we can see that a new child process is forked from the parent process and the parent process is terminated.
- After the parent process terminates, the child process prints “child process” and after 15 seconds, it has printed the statement given after the sleep() statement.
- Since, the parent process is already terminated, the process which prints the after statement, is called an orphan process.

5. **How would you theorize a Zombie process? How do we find a zombie process through the ‘ps’ command? Write a C Program to create a Zombie process**

   A Zombie process is a process that has completed execution but still has an entry in the process table. This typically happens when the parent process has not yet read the exit status of the terminated child process. Every child process sends a SIGCHLD signal to its parent upon termination, and the parent process is supposed to call one of the wait system calls like wait() to read the exit status of the child. If the parent process does not handle this signal or read the exit status, the child process remains in the process table as a Zombie.

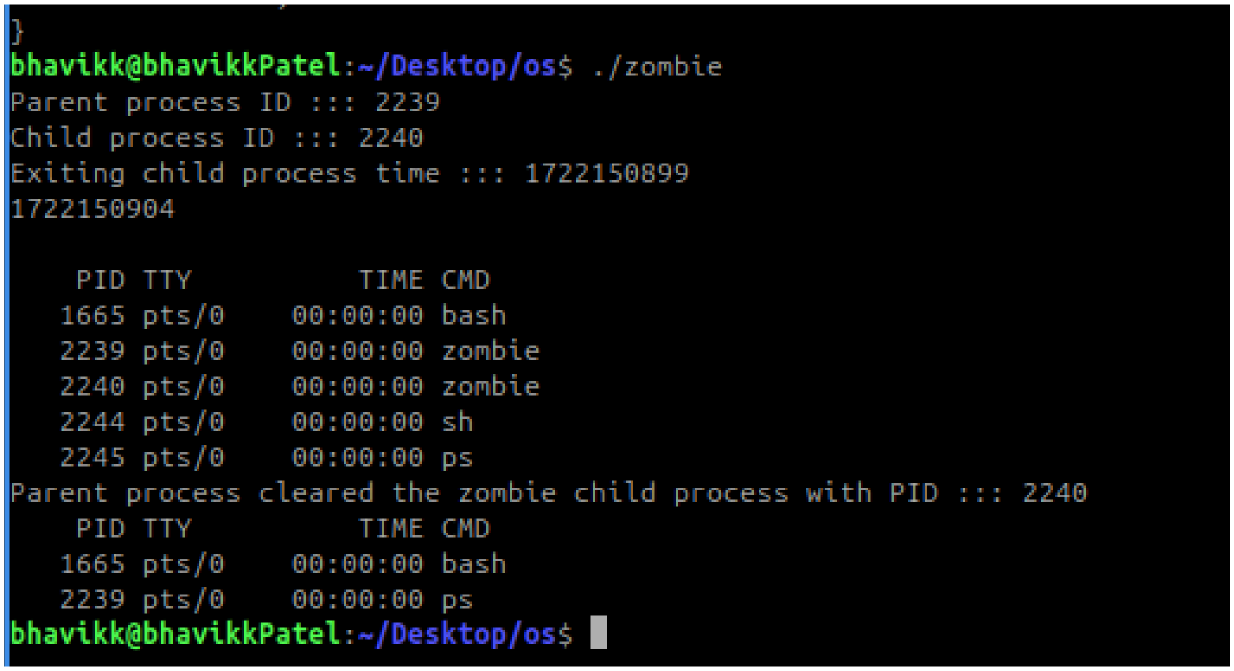
   In the below code we can visualize the zombie process as the parent calls the wait() after a sleep of 10 seconds, but the child is executed with exit() to terminate it.

```
#include <iostream>
#include <unistd.h>
#include <ctime>
#include <sys/types.h>
#include <sys/wait.h>
#include <cstdlib>
// Bhavikk Patel
// 24MCS0047
int main () {
        pid_t pid = fork();
        if (pid > 0) {
        std::cout << "Parent process ID ::: " << getpid() << std::endl;
        std::cout << "Child process ID ::: " << pid  << std::endl;
        sleep(5);
        std::cout << std::time(0) << std::endl;
        std::string cmd = "ps -p " + pid;
        std::cout << cmd.c_str() << std::endl;
        system("ps");
        wait(NULL);
        std::cout << "Parent process cleared the zombie child process with PID ::: " << pid
<< std::endl;
        execl("/bin/ps", "-p " + pid, (char *)0);
        } else if (pid == 0) {
        std::cout << "Exiting child process" << " time ::: " << std::time(0)  << std::endl;
        exit(0);
        } else {
        std::cout << "Failed" << std::endl;
        return 1;
        }
        return 0;
}
```


bhavikk@bhavikkPatel: ~/Desktop/os
File Actions Edit View Help
bhavikk@bhavikkPatel: ~/Desktop/os

```
#include <cstdlib>

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

        if (pid > 0) {
                std::cout << "Parent process ID ::: " << getpid() << std::endl;
                std::cout << "Child process ID ::: " << pid  << std::endl;
                sleep(5);
                std::cout << std::time(0) << std::endl;
                std::string cmd = "ps -p " + pid;
                std::cout << cmd.c_str() << std::endl;
                system("ps");
                wait(NULL);
                std::cout << "Parent process cleared the zombie child process with PID ::: " << pid << std::endl;
                execl("/bin/ps", "-p " + pid, (char *)0);
        } else if (pid == 0) {
                std::cout << "Exiting child process" << " time ::: " << std::time(0)  << std::endl;
                exit(0);
        } else {
                std::cout << "Failed" << std::endl;
                return 1;
        }
        return 0;
}
bhavikk@bhavikkPatel:~/Desktop/os$ ./zombie
Parent process ID ::: 2239
Child process ID ::: 2240
Exiting child process time ::: 1722150899
1722150904

    PID TTY          TIME CMD
   1665 pts/0    00:00:00 bash
   2239 pts/0    00:00:00 zombie
   2240 pts/0    00:00:00 zombie
   2244 pts/0    00:00:00 sh
   2245 pts/0    00:00:00 ps
Parent process cleared the zombie child process with PID ::: 2240
    PID TTY          TIME CMD
   1665 pts/0    00:00:00 bash
   2239 pts/0    00:00:00 ps
bhavikk@bhavikkPatel:~/Desktop/os$
```

- During the sleep period of the parent process, we can observe the zombie process using the ps command as shown in the output.
- The output will show the child process with a status 'Zombie'.
- Once the parent process reaps the child, the zombie process will be removed from the process table, and it will no longer appear in the ps output.



```
}
bhavikk@bhavikkPatel:~/Desktop/os$ ./zombie
Parent process ID ::: 2239
Child process ID ::: 2240
Exiting child process time ::: 1722150899
1722150904


    PID TTY          TIME CMD
   1665 pts/0    00:00:00 bash
   2239 pts/0    00:00:00 zombie
   2240 pts/0    00:00:00 zombie
   2244 pts/0    00:00:00 sh
   2245 pts/0    00:00:00 ps
Parent process cleared the zombie child process with PID ::: 2240
    PID TTY          TIME CMD
   1665 pts/0    00:00:00 bash
   2239 pts/0    00:00:00 ps
bhavikk@bhavikkPatel:~/Desktop/os$
```