**Commands**

* apt -get update && apt -get upgrade
* uname
* uname -r
* uname -a
* history **for listing recently used commands**
* clear **for clear the terminal**
* ls **for list Directory**
* sana@sana/$  **to access user directory**
* sana@sana~$  **to access root directory**
* cd(catch Directory)  **to get back from user/root dirctory**
* touch filename **to create file**
* rm filename **to remove file**
* mkdir folder name  **to make folder**
* rmdir folder name **to del folder**

**Permission**

**Admin, group, public**

**7 6 5**

**excute (1) x**

**write (2) w**

**Read (4) r**

ls -l filename.ext **to check permission**

**Result:** -rw-rw-r-- 1

-rw rw- r--

chmod 765 filename.ext **to change your file permission**

**Result**

-rwxrw-r-xcd

1. **To write c Program for system call in terminal**

nano first.c  **to write c code**

**we reached GNU platform**

#include<unistd.h>

int main()

{

write(1, "hello",5);

}

Ctrl+s **for save code**

ctrl+x **to exit from GNU platform**

**we reached back to terminal**

gcc first.c **to execute c code**

./a.out **to execute object file of c code**

1. **Write c program for get Process id**

nano fork.c  **to write c code**

**we reached GNU platform**

#include<unistd.h>

int main()

{

Printf(PID of running process=%d\n”, getpid());

Return 0;

}

Ctrl+s **for save code**

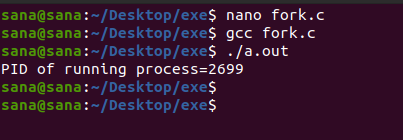
ctrl+x **to exit from GNU platform**

**we reached back to terminal**

gcc fork.c **to execute c code**

./exe1.c **to execute object file of c code**

**Output:**



1. **Write c program for fork system call**

**Fork():**

Fork() system call is used to create separate/duplicate/ child process with same content of file but assign different PIDs

nano fork.c  **to write c code**

**we reached GNU platform**

#include<unistd.h>

int main()

{

Fork();

Printf(PID of running process=%d\n”, getpid());

Return 0;

}

Ctrl+s **for save code**

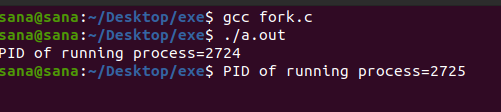
ctrl+x **to exit from GNU platform**

**we reached back to terminal**

gcc fork.c **to execute c code**

./exe1.c **to execute object file of c code**

**Output:**



1. **Write c program for multiple fork system call**

nano fork.c  **to write c code**

**we reached GNU platform**

#include<unistd.h>

int main()

{

Fork();

Fork();

Fork();

Printf(PID of running process=%d\n”, getpid());

Return 0;

}

Ctrl+s **for save code**

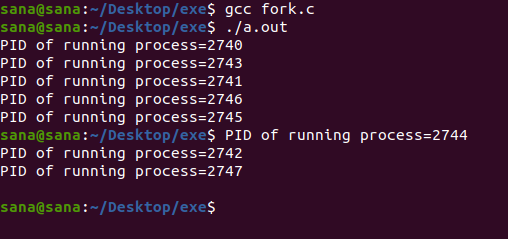
ctrl+x **to exit from GNU platform**

**we reached back to terminal**

gcc fork.c **to execute c code**

./exe1.c **to execute object file of c code**

**Output:**



1. **Write c program for exec system call**

**Exec() system call:**

Exec() system call is used to create separate/duplicate/ child process with different content of file but assign same PIDs

nano exe2.c

**we reached GNU platform**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

int main(int argc, char \*argv[])

{

Printf(“we are in exe2.c\n”);

Printf(PID of exe2.c=%d\n”, getpid());

Return 0;

}

nano exe1.c

**we reached GNU platform**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

int main(int argc, char \*argv[])

{

Printf(PID of exe1.c=%d\n”, getpid());

Char \* args={“hello”, “programmer”,null};

Execv(“./exe2.c”, args);

Printf(“back to exe1.c”);

Return 0;

}

Ctrl+s **for save code**

ctrl+x **to exit from GNU platform**

**we reached back to terminal**

gcc exe1.c -o exe1 **to execute c code**

gcc exe2.c -o exe2 **to execute c code**

./exe1 **to execute object file of c code**

**Output:**

