Name: Brijesh Rohit

Admission no.: U19CS009

SYSTEM SOFTWARES - ASSIGNMENT -1

Fork and Getpid:

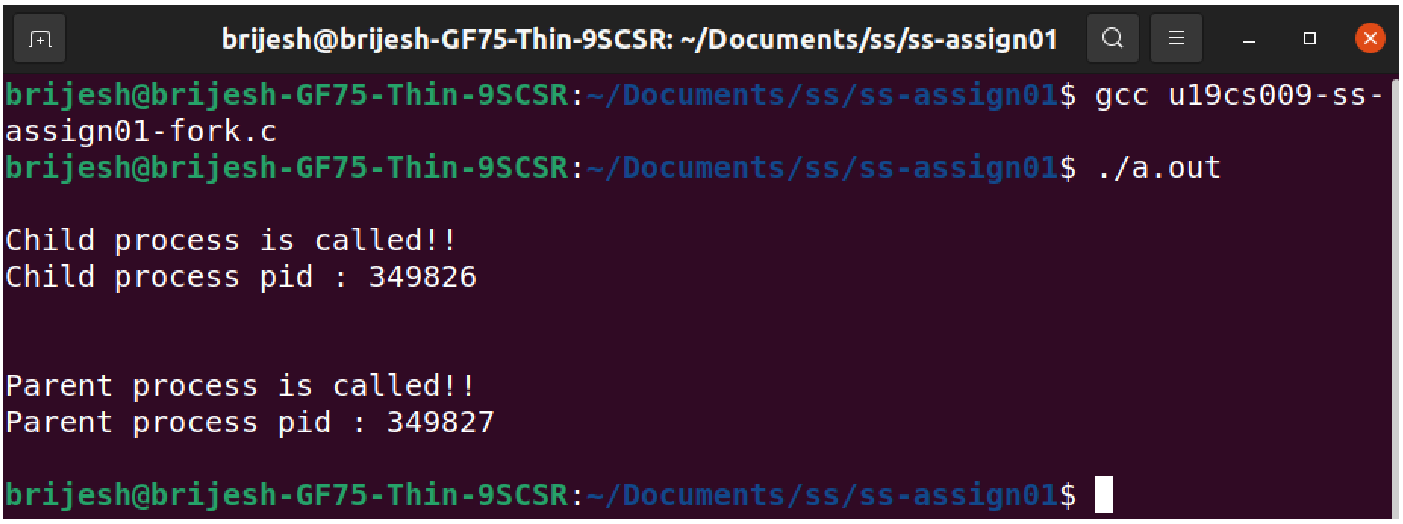
fork() creates a new child process of the previous parent process.

getpid() returns the process id of the calling function.

Code=>

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

int main()
{
    if (fork() == 0)
    {
        printf("\nParent process is called!!");
        printf("\nParent process pid : %d\n", getpid());
    }
    else
    {
        printf("\nChild process is called!!");
        printf("\nChild process pid : %d\n", getpid());
    }
    printf("\n");
    return 0;
}
```

Output=>



```
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ gcc u19cs009-ss-
assign01-fork.c
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ ./a.out

Child process is called!!
Child process pid : 349826


Parent process is called!!
Parent process pid : 349827

brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ 
```

Exec:

It creates and replaces the currently running process with another process

Code=>

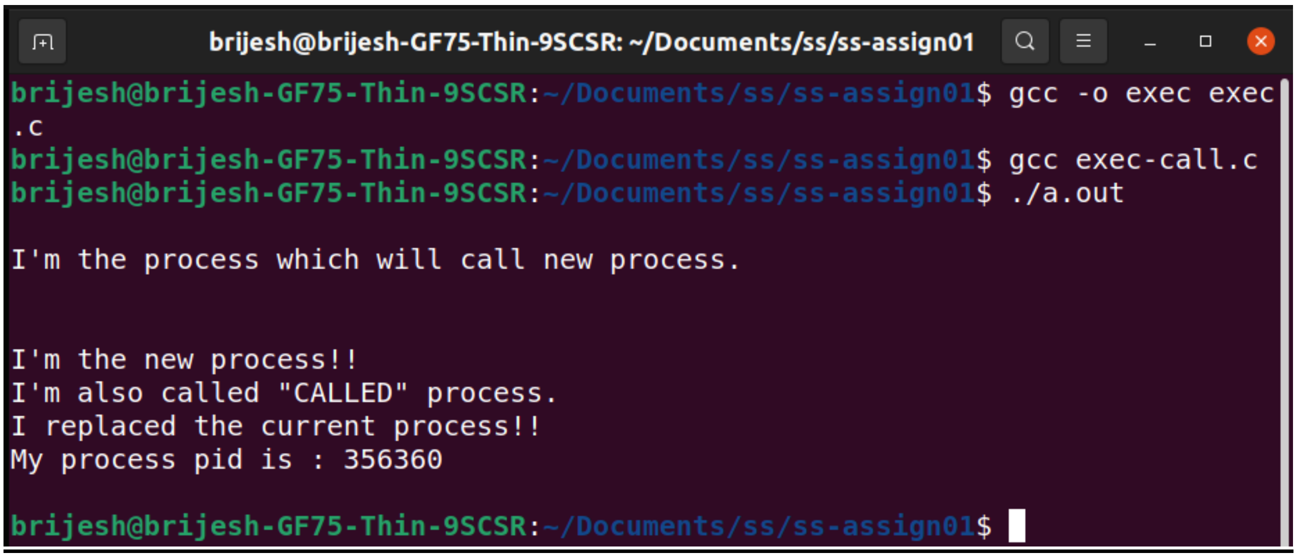
exec.c [called file]

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

int main()
{
    //EXEC
    printf("\n\nI'm the new process!!\nI'm also called \"CALLED\" process.");
    printf("\nI replaced the current process!!");
    printf("\nMy process pid is : %d", getpid());
    printf("\n\n");
    return 0;
}
```

exec-call.c [calling file]

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

int main()
{
    printf("\nI'm the process which will call new process.");
    printf("\nI'm also called \"CALLING\" process.");
    //EXEC
    char *args[] = {"./exec", NULL};
    execvp(args[0], args);
    printf("\nAfter calling, I'm running a print statement.");
    printf("\nWhich will not be printed.\n");
    return 0;
}
```

Output=>



brijesh@brijesh-GF75-Thin-9SCSR: ~/Documents/ss/ss-assign01

```
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ gcc -o exec exec
.c
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ gcc exec-call.c
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ ./a.out

I'm the process which will call new process.


I'm the new process!!
I'm also called "CALLED" process.
I replaced the current process!!
My process pid is : 356360

brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ 
```

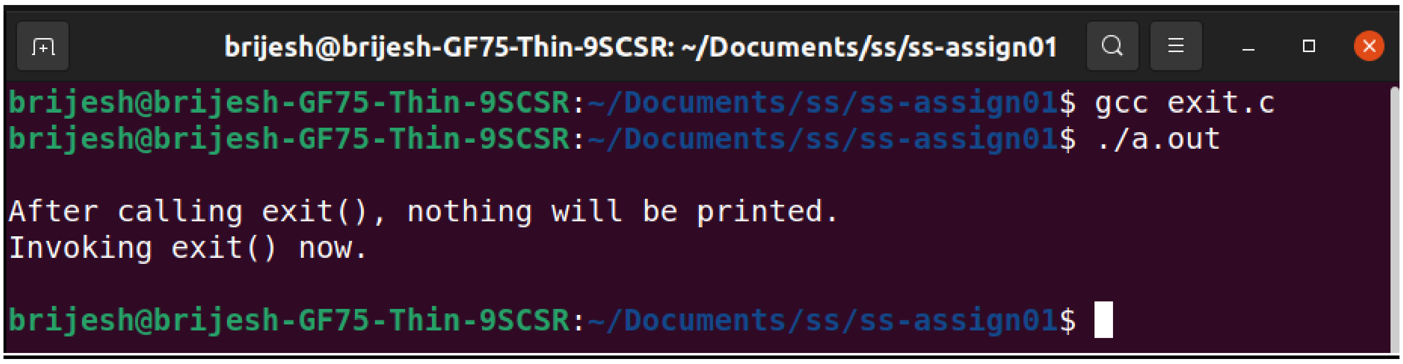
Exit:

Terminates the execution of a program

CODE=>

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

int main()
{
    printf("\nAfter calling exit(), nothing will be printed.");
    printf("\nInvoking exit() now.\n\n");
    exit(0);
    printf("not printing because of exit()");
    return 0;
}
```

Output=>



```
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ gcc exit.c
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ ./a.out

After calling exit(), nothing will be printed.
Invoking exit() now.

brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ 
```

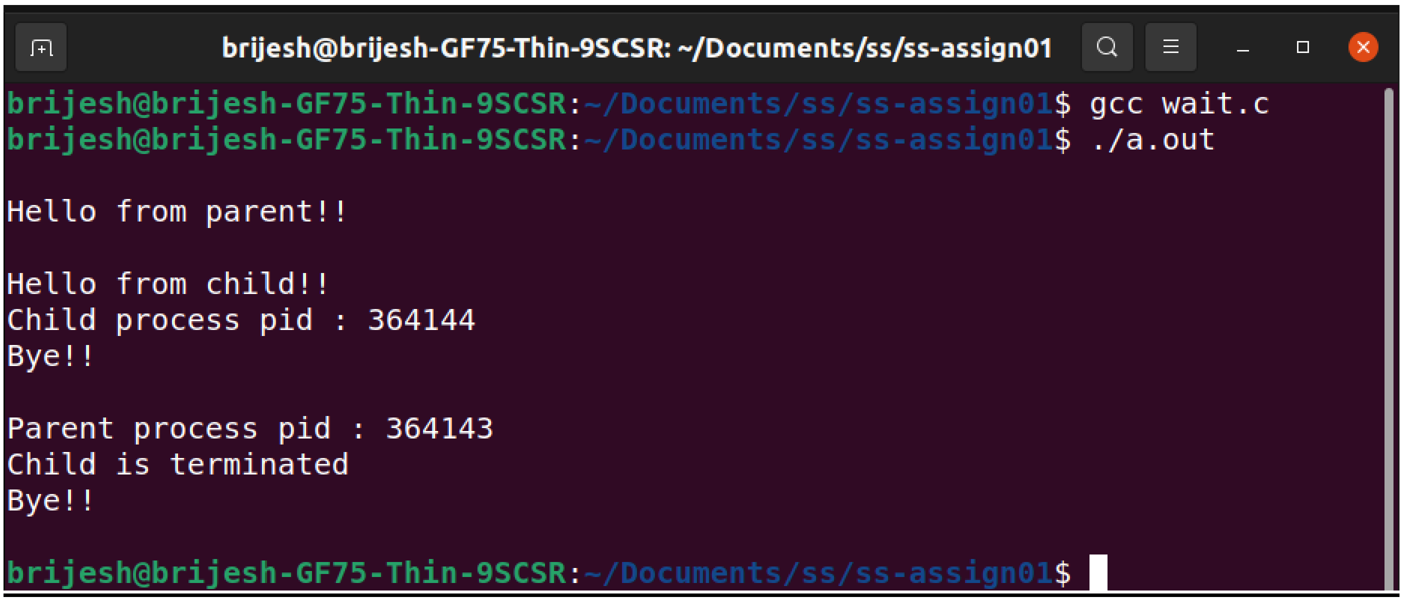
Wait:

Wait() blocks the execution of parent process until a child process has finished executing or a signal is received. After child process ends parent will continue its execution.

CODE=>

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

int main() {
    int cpid;
    if (fork()== 0)
    {
        printf("\nHello from child!!");
        printf("\nChild process pid : %d", getpid());
    }
    else
    {
        printf("\nHello from parent!!");
        printf("\nParent process pid : %d", getpid());
        wait(NULL);
        printf("\nChild is terminated");
    }
    printf("\nBye!!");
    printf("\n\n");
    return 0;
}
```

Output=>

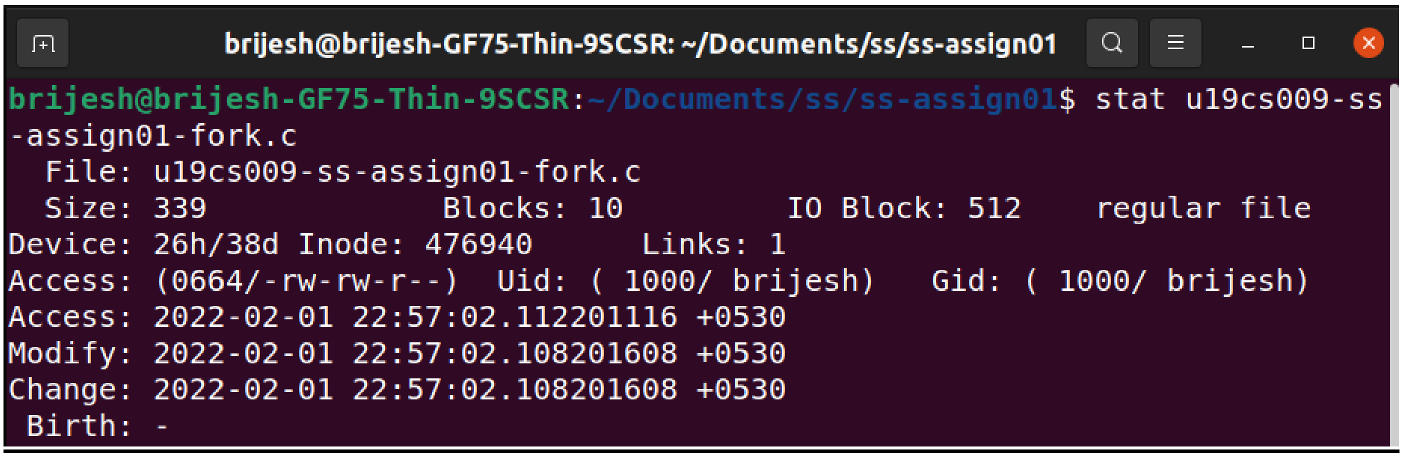


```
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ gcc wait.c
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ ./a.out

Hello from parent!!

Hello from child!!
Child process pid : 364144
Bye!!

Parent process pid : 364143
Child is terminated
Bye!!

brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ 
```

Stat:

It is used to display the status of a file (size, name, blocks used, etc.).

Output=>



```
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ stat u19cs009-ss
-assign01-fork.c
  File: u19cs009-ss-assign01-fork.c
  Size: 339             Blocks: 10         IO Block: 512    regular file
Device: 26h/38d Inode: 476940      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/ brijesh)   Gid: ( 1000/ brijesh)
Access: 2022-02-01 22:57:02.112201116 +0530
Modify: 2022-02-01 22:57:02.108201608 +0530
Change: 2022-02-01 22:57:02.108201608 +0530
 Birth: -
```

Opendir:

It is used to open a directory stream corresponding to the directory name.

Closedir:

It closes the currently opened directory stream.

Readdir:

Reads the files and directories present in the opened directory stream.

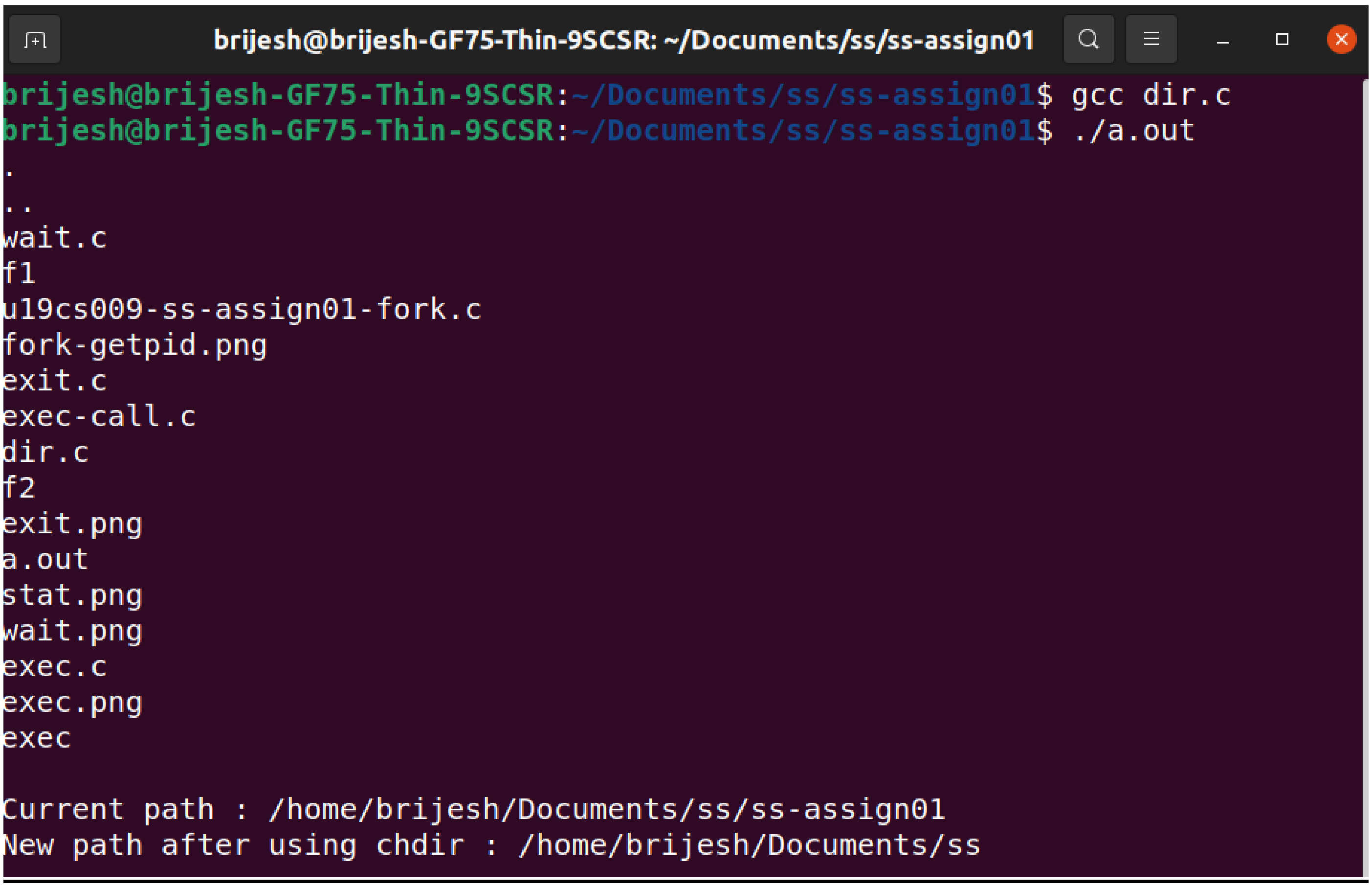
Chdir:

It changes the current working directory to another one specified by user.

CODE=>

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <dirent.h>
#include <errno.h>

int main()
{
    struct dirent *dir;
    DIR* d = opendir(".");
    //READDIR
    while ((dir = readdir(d)) != NULL)
        printf("%s\n", dir->d_name);

    char s1[100];

    printf("\nCurrent path : %s\n", getcwd(s1, 50)); //CHDIR
    chdir("..");

    printf("New path after using chdir : %s\n", getcwd(s1, 50)); //READDIR
    while ((dir = readdir(d)) != NULL)
        printf("%s\n", dir->d_name);

    //CLOSEDIR
    closedir(d);
    printf("\n\n");
    exit(EXIT_SUCCESS);
    return 0;
}
```

Output=>

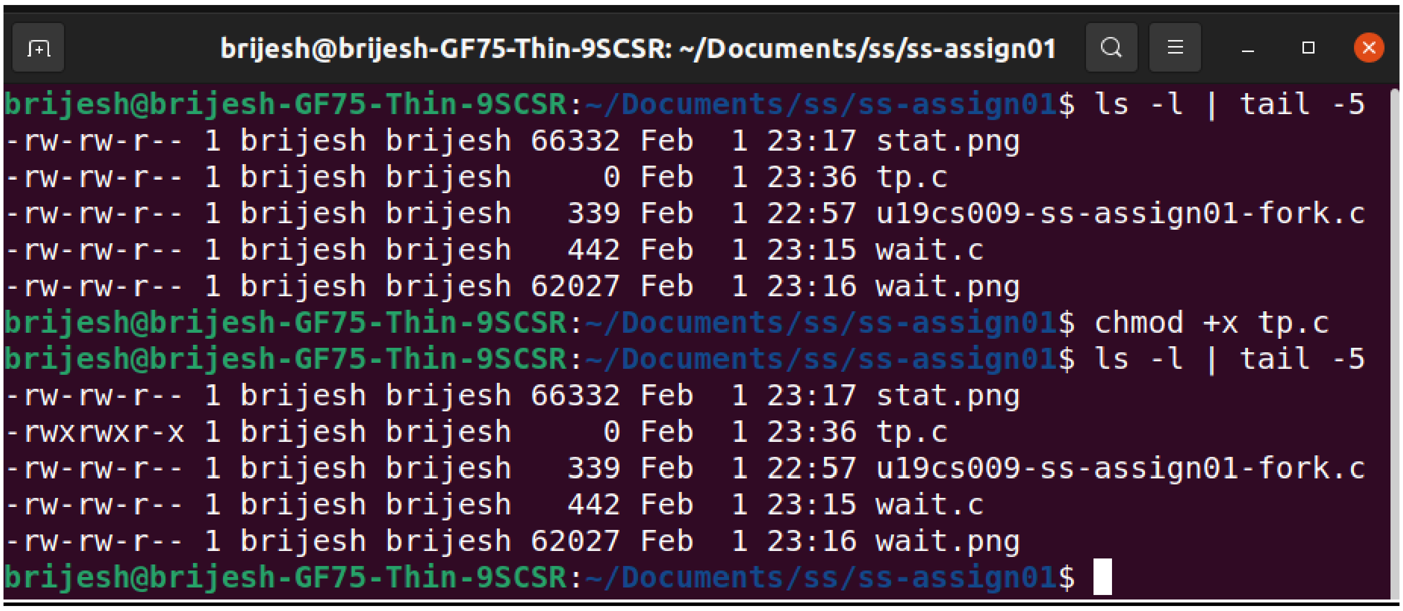


```
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ gcc dir.c
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ ./a.out
.
..
wait.c
f1
u19cs009-ss-assign01-fork.c
fork-getpid.png
exit.c
exec-call.c
dir.c
f2
exit.png
a.out
stat.png
wait.png
exec.c
exec.png
exec

Current path : /home/brijesh/Documents/ss/ss-assign01
New path after using chdir : /home/brijesh/Documents/ss
```

Chmod:

It is used change the access permissions or modes of a file-system.

OUTPUT=>



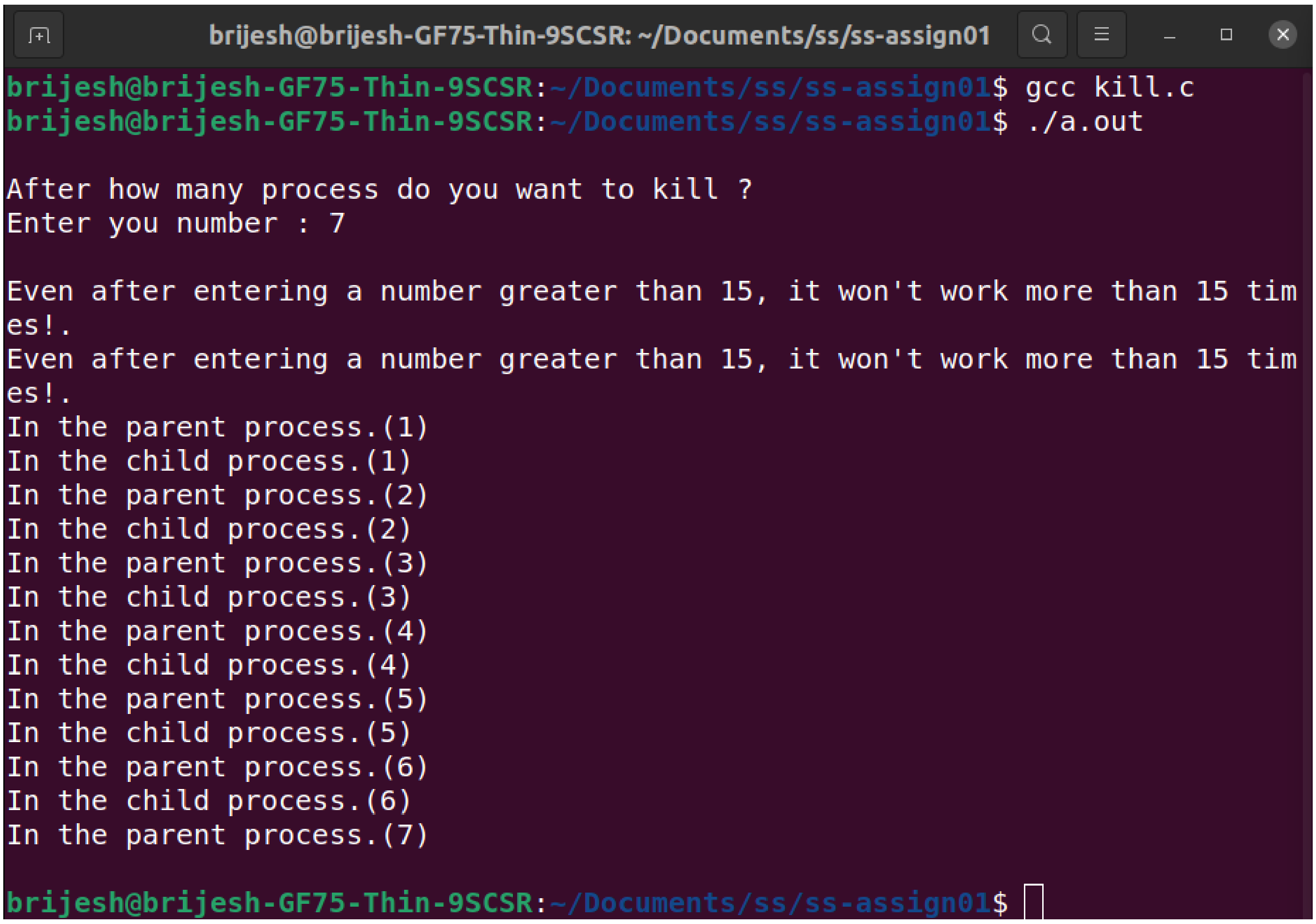
```
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ ls -l | tail -5
-rw-rw-r-- 1 brijesh brijesh 66332 Feb  1 23:17 stat.png
-rw-rw-r-- 1 brijesh brijesh     0 Feb  1 23:36 tp.c
-rw-rw-r-- 1 brijesh brijesh   339 Feb  1 22:57 u19cs009-ss-assign01-fork.c
-rw-rw-r-- 1 brijesh brijesh   442 Feb  1 23:15 wait.c
-rw-rw-r-- 1 brijesh brijesh 62027 Feb  1 23:16 wait.png
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ chmod +x tp.c
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ ls -l | tail -5
-rw-rw-r-- 1 brijesh brijesh 66332 Feb  1 23:17 stat.png
-rwxrwxr-x 1 brijesh brijesh     0 Feb  1 23:36 tp.c
-rw-rw-r-- 1 brijesh brijesh   339 Feb  1 22:57 u19cs009-ss-assign01-fork.c
-rw-rw-r-- 1 brijesh brijesh   442 Feb  1 23:15 wait.c
-rw-rw-r-- 1 brijesh brijesh 62027 Feb  1 23:16 wait.png
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$
```

Kill:

CODE=>

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

int main()
{
    printf("\nAfter how many process do you want to kill ?");
    printf("\nEnter you number : ");
    int n;
    scanf("%d",&n);
    printf("\nEven after entering a number greater than 15, it won't work more than
15 times!.");

    pid_t retVal;
    retVal = fork();
    if(retVal > 0)
    {
        int i = 0;
        while(i++ < n)
        {
            printf("\nIn the parent process.(%d)",i);
            sleep(1);
        }
        //kill the child process
        kill(retVal, SIGKILL);

    }
    else if (retVal == 0)
    {
        int i = 0;
        //will not ever get to 15, because
        //the parent process will kill it
        while(i++ < 15)
        {
            printf("\nIn the child process.(%d)", i);
            sleep(1);
        }
    }
    else
    {
        //something bad happened.
        printf("\nSomething bad happened.");
        exit(EXIT_FAILURE);
    }
    printf("\n\n");
    return 0;
}
```

Output=>



```
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ gcc kill.c
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ ./a.out

After how many process do you want to kill ?
Enter you number : 7

Even after entering a number greater than 15, it won't work more than 15 tim
es!.
Even after entering a number greater than 15, it won't work more than 15 tim
es!.
In the parent process.(1)
In the child process.(1)
In the parent process.(2)
In the child process.(2)
In the parent process.(3)
In the child process.(3)
In the parent process.(4)
In the child process.(4)
In the parent process.(5)
In the child process.(5)
In the parent process.(6)
In the child process.(6)
In the parent process.(7)

brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ 
```

Read:

It reads specified bytes of input indicated by the file descriptor in to the memory buffer.

Write:

It writes specified bytes from the memory buffer to the file indicated by the file descriptor.

Open:

Opens a file for reading, writing or both.

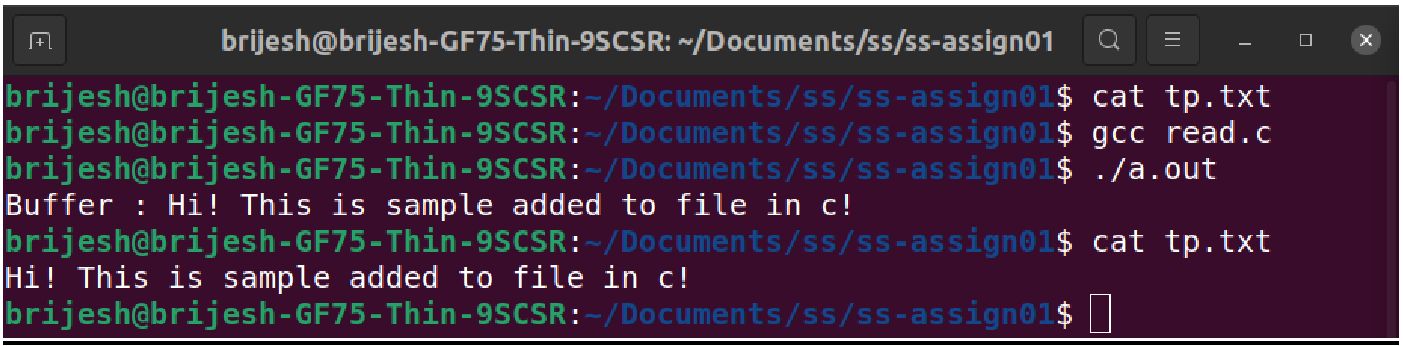
Close:

Closes the file opened indicated by the file descriptor.

CODE=>

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <dirent.h>
#include <errno.h>

int main()
{
    int f;
    char buffer[100];
    //OPEN
    f = open("tp.txt", O_CREAT | O_WRONLY, 0600);
    if (f == -1)
    {
        printf("Error opening the file");
        exit(1);
    }
    //WRITE CLOSE
    write(f, "Hi! This is sample added to file in c!\n", 39);
    close(f);
    //READ
    f = open("tp.txt", O_RDONLY);
    if (f == -1)
    {
        printf("Error opening the file");
        exit(1);
    }
    read(f, buffer, 39);
    buffer[39] = '\0';
    close(f);
    printf("Buffer : %s", buffer);
    return 0;
}
```

Output=>



```
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ cat tp.txt
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ gcc read.c
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ ./a.out
Buffer : Hi! This is sample added to file in c!
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ cat tp.txt
Hi! This is sample added to file in c!
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ 
```

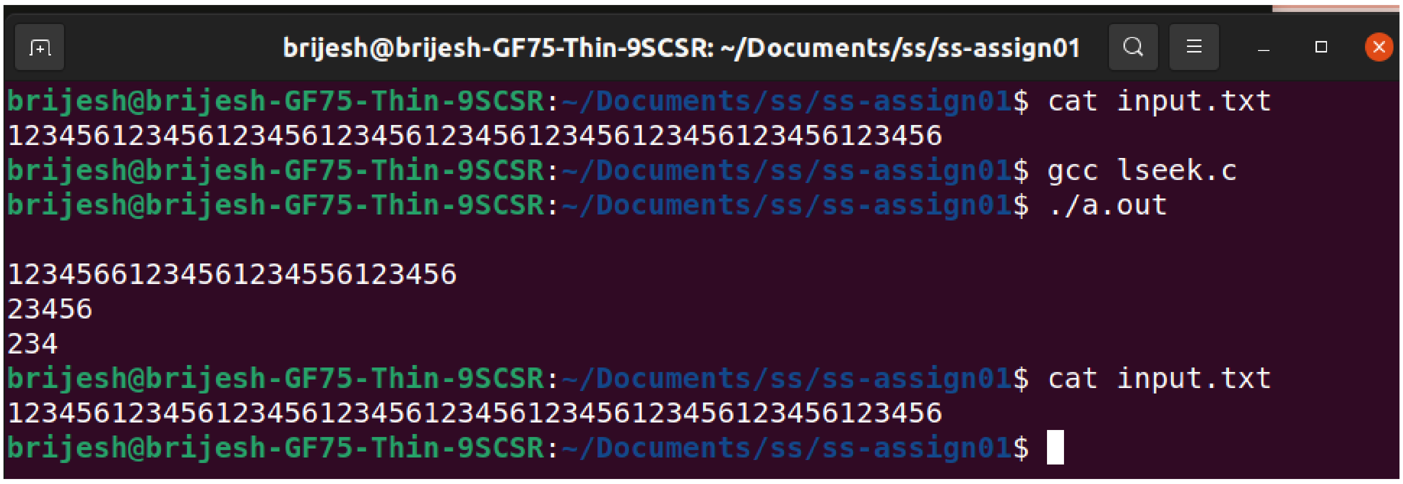
Lseek:

Used to change the read/write pointer of a file descriptor.

CODE=>

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <dirent.h>
#include <errno.h>

int main()
{
    printf("\n");
    int n, f;
    char buff[6];
    f = open("input.txt",O_RDWR);
    while(read(f,buff,6))
    {
        read(f,buff,6);
        write(1,buff,6);
        lseek(f,5,SEEK_CUR);
        read(f,buff,6);
        write(1,buff,6);
    }
    printf("\n");
    return 0;
}
```

Output=>

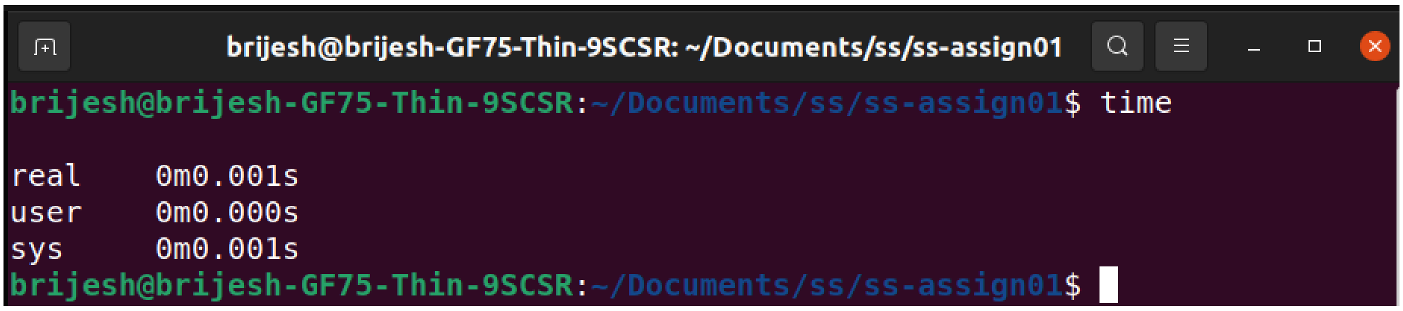


```
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ cat input.txt
123456123456123456123456123456123456123456123456123456
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ gcc lseek.c
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ ./a.out

12345661234561234556123456
23456
234
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ cat input.txt
123456123456123456123456123456123456123456123456123456
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ 
```

Time:

Returns the time since epoch.

Output=>



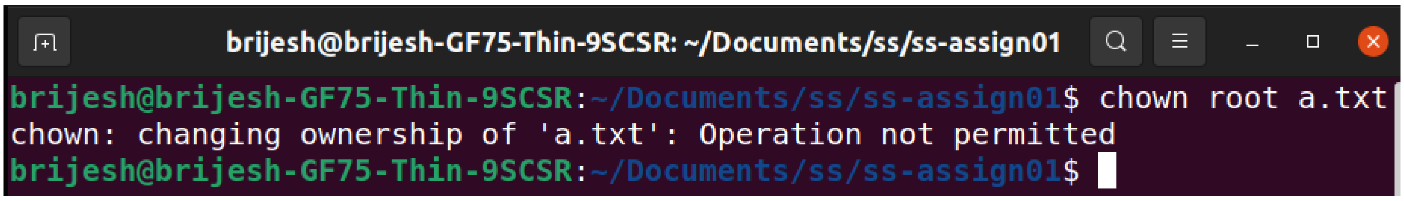
```
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ time

real	0m0.001s
user	0m0.000s
sys	0m0.001s
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ 
```

Mount:

It is used to mount the file-system found on a device to big tree structure(Linux file-system) rooted at '/'.

Chown:

It is used to change the owner and group of the file specified by the file descriptor or path.



```
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ chown root a.txt
chown: changing ownership of 'a.txt': Operation not permitted
brijesh@brijesh-GF75-Thin-9SCSR:~/Documents/ss/ss-assign01$ 
```