

1) To study the basics of system call and system library.

A system call is the method through which a computer programme asks a service from the kernel of the operating system on which it is running. A system call is a method of interacting with the operating system via programmes. When a computer software makes a request to the kernel of the operating system, it is called a system call. The operating system's services are provided to user programmes via the Application Program Interface (API)

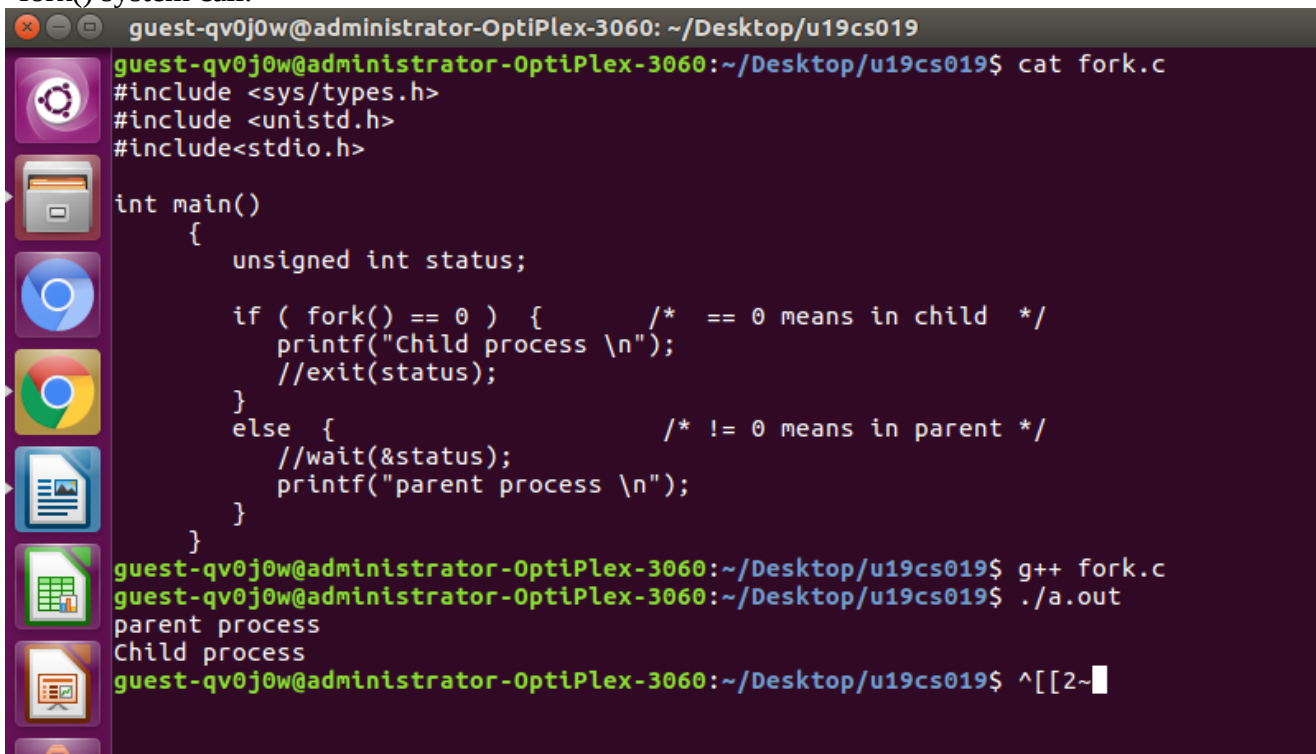
It acts as a link between a process and the operating system, allowing user-level programmes to request operating system services. The kernel system can only be accessed using system calls. System calls are required for any programmes that use resources.

2) Study the following system calls :

Fork, exec, getpid, exit, wait, stat, opendir, readdir, chdir, chmod, kill, read, write, open, close, lseek, time, mount, chown.

FORK

Fork system call is used for creating a new process, which is called child process, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call.



```
guest-qv0j0w@administrator-OptiPlex-3060: ~/Desktop/u19cs019
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ cat fork.c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

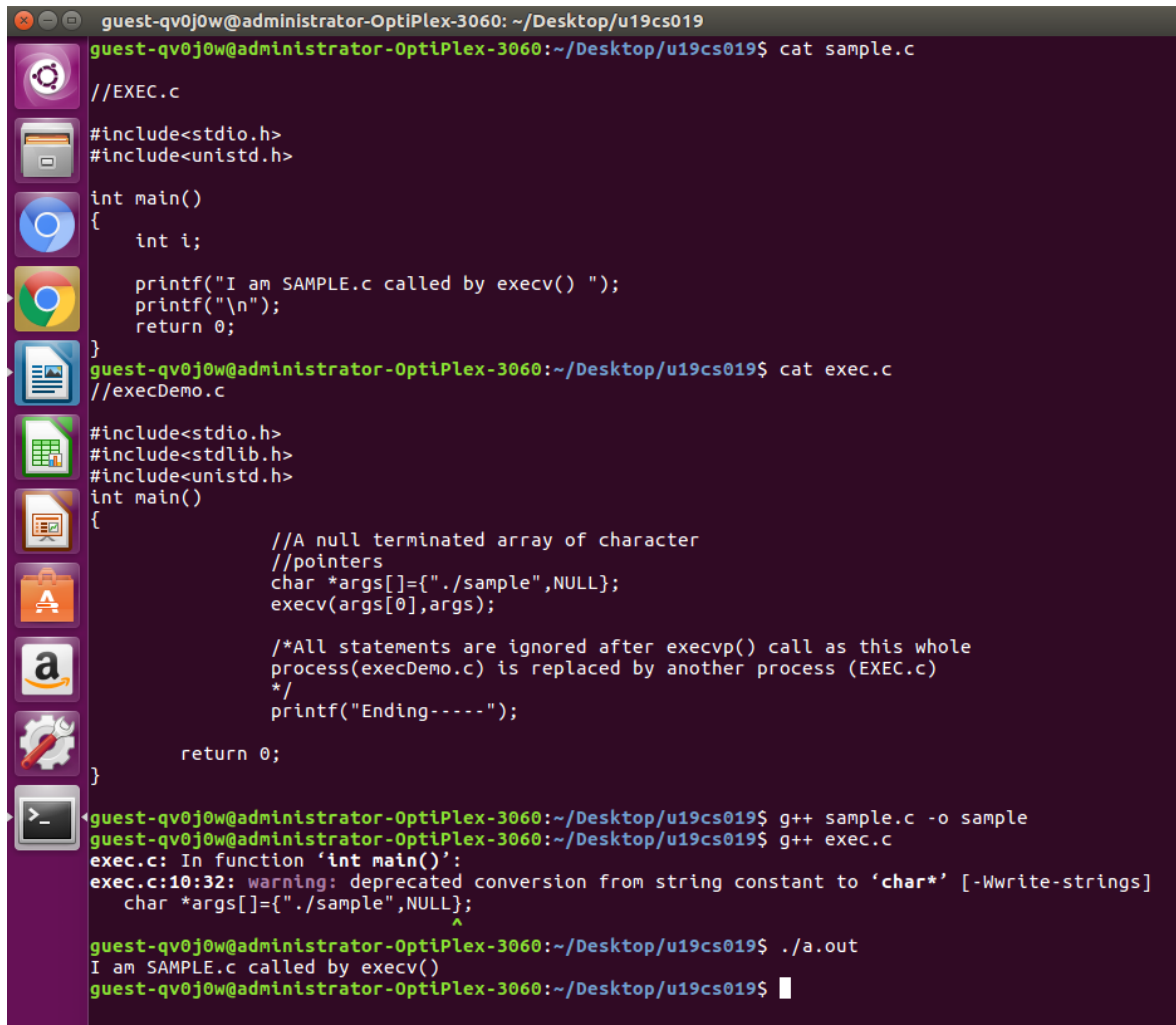
int main()
{
    unsigned int status;

    if ( fork() == 0 ) {          /* == 0 means in child */
        printf("Child process \n");
        //exit(status);
    }
    else {                      /* != 0 means in parent */
        //wait(&status);
        printf("parent process \n");
    }
}

guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ g++ fork.c
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ ./a.out
parent process
Child process
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ ^[[2~
```

EXEC

The exec family of functions replaces the current running process with a new process. It can be used to run a C program by using another C program.



```
guest-qv0j0w@administrator-OptiPlex-3060: ~/Desktop/u19cs019
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ cat sample.c
//EXEC.c
#include<stdio.h>
#include<unistd.h>

int main()
{
    int i;

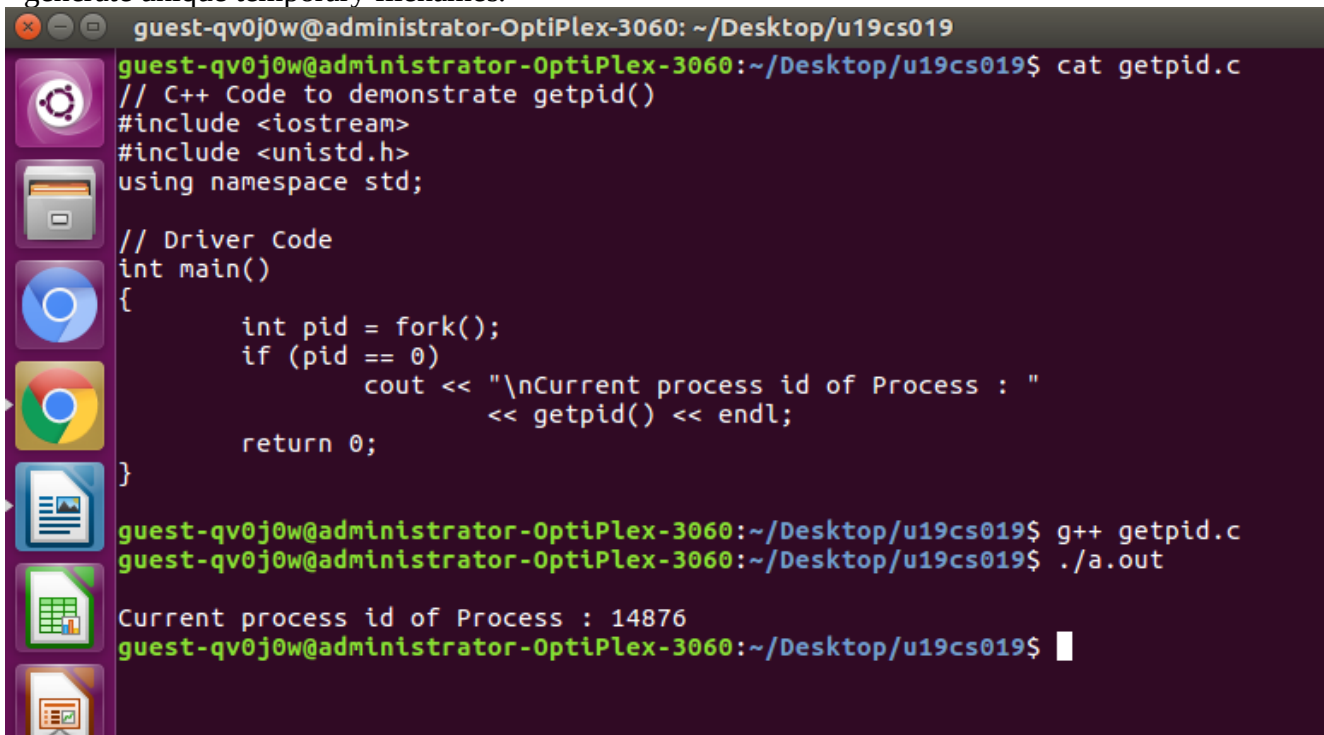
    printf("I am SAMPLE.c called by execv() ");
    printf("\n");
    return 0;
}
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ cat exec.c
//execDemo.c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
    //A null terminated array of character
    //pointers
    char *args[]={"/.sample",NULL};
    execv(args[0],args);

    /*All statements are ignored after execvp() call as this whole
    process(execDemo.c) is replaced by another process (EXEC.c)
    */
    printf("Ending-----");

    return 0;
}
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ g++ sample.c -o sample
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ g++ exec.c
exec.c: In function 'int main()':
exec.c:10:32: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
    char *args[]={"/.sample",NULL};
                           ^
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ ./a.out
I am SAMPLE.c called by execv()
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$
```

GETPID

getpid() returns the process ID of the calling process. This is often used by routines that generate unique temporary filenames.



```
guest-qv0j0w@administrator-OptiPlex-3060: ~/Desktop/u19cs019
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ cat getpid.c
// C++ Code to demonstrate getpid()
#include <iostream>
#include <unistd.h>
using namespace std;

// Driver Code
int main()
{
    int pid = fork();
    if (pid == 0)
        cout << "\nCurrent process id of Process : "
              << getpid() << endl;

    return 0;
}

guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ g++ getpid.c
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ ./a.out

Current process id of Process : 14876
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$
```

EXIT

exit() terminates the calling process immediately.



```
guest-qv0j0w@administrator-OptiPlex-3060: ~/Desktop/u19cs019
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ cat exit.c
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    // declaration of the variables
    for (int i = 1; i<15; i++)
    {
        // use if statement to check the condition
        if ( i == 6 )

            /* use exit () statement with passing 0 argument to show termination of the program without any error message. */
            exit(0);

        else

            printf ("\tNumber is %d\n", i);
    }
    return 0;
}

guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ g++ exit.c
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ ./a.out
Number is 1
Number is 2
Number is 3
Number is 4
Number is 5
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$
```

STAT

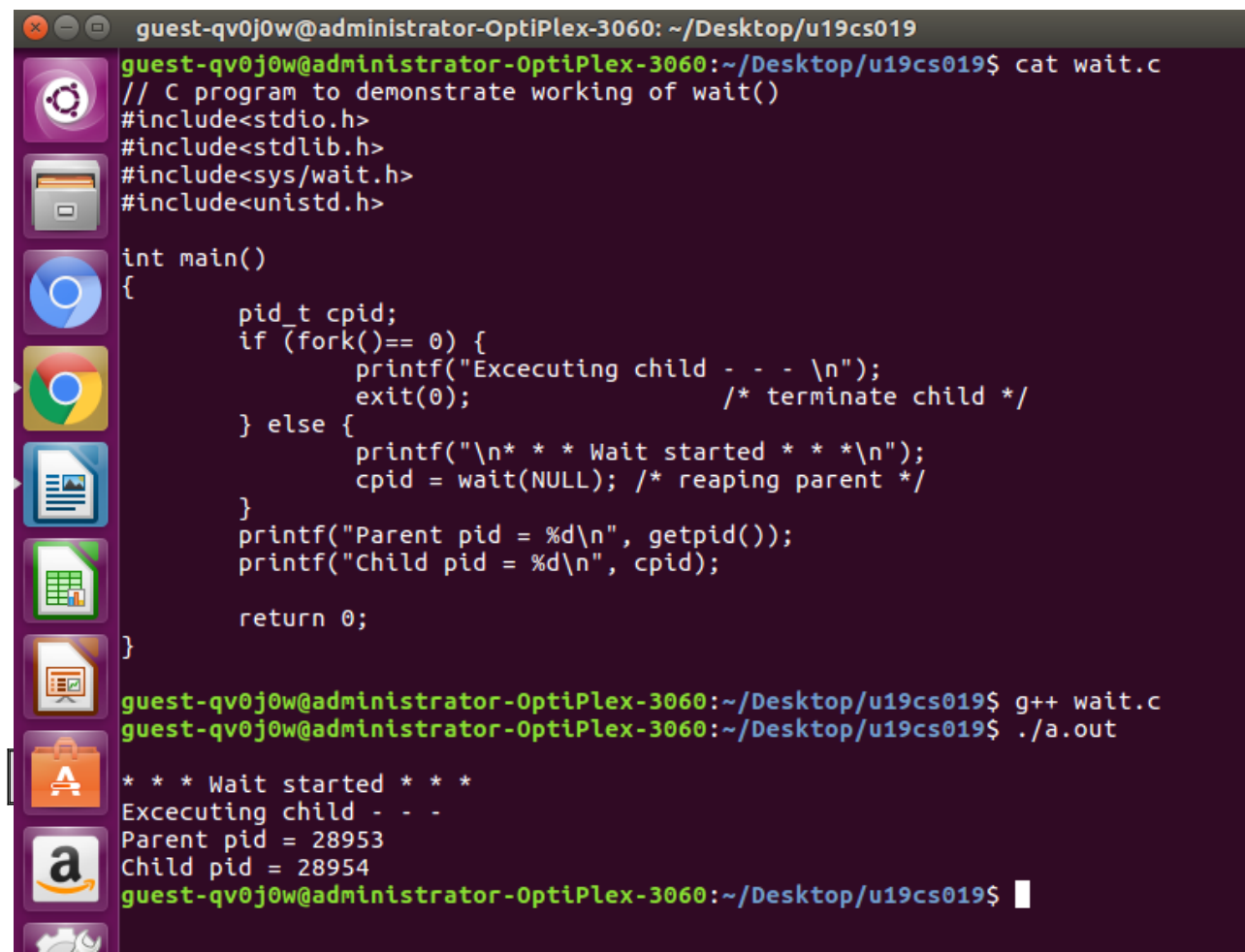
Stat command gives information such as the size of the file, access permissions and the user ID and group ID, birth time access time of the file. Stat command has another feature, by which it can also provide the file system information.



```
guest-qv0j0w@administrator-OptiPlex-3060: ~/Desktop/u19cs019
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ stat sample.c
  File: 'sample.c'
  Size: 166             Blocks: 8          IO Block: 4096   regular file
Device: 31h/49d Inode: 2561         Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 984/guest-qv0j0w)   Gid: ( 984/guest-qv0j0w)
Access: 2022-01-06 14:41:55.819220786 +0530
Modify: 2022-01-06 14:41:50.872328150 +0530
Change: 2022-01-06 14:41:50.872328150 +0530
 Birth: -
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$
```

WAIT

A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait system call instruction.



```
guest-qv0j0w@administrator-OptiPlex-3060: ~/Desktop/u19cs019
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ cat wait.c
// 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) {
        printf("Excecuting child - - - \n");
        exit(0);                /* terminate child */
    } else {
        printf("\n* * * Wait started * * *\n");
        cpid = wait(NULL); /* reaping parent */
    }
    printf("Parent pid = %d\n", getpid());
    printf("Child pid = %d\n", cpid);

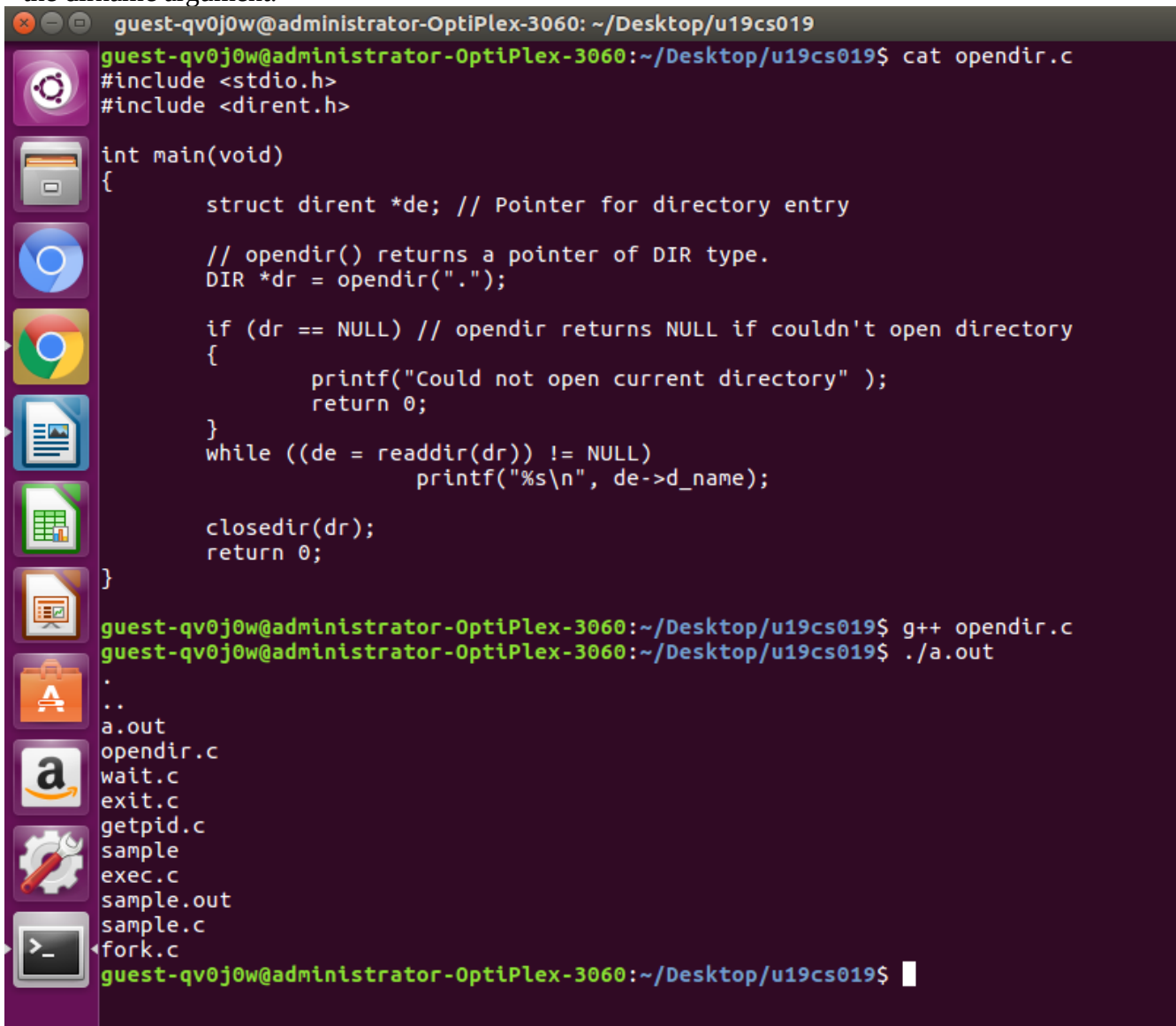
    return 0;
}

guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ g++ wait.c
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ ./a.out

* * * Wait started * * *
Excecuting child - - -
Parent pid = 28953
Child pid = 28954
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$
```

OPENDIR

The opendir() function shall open a directory stream corresponding to the directory named by the dirname argument.



```
guest-qv0j0w@administrator-OptiPlex-3060: ~/Desktop/u19cs019
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ cat opendir.c
#include <stdio.h>
#include <dirent.h>

int main(void)
{
    struct dirent *de; // Pointer for directory entry

    // opendir() returns a pointer of DIR type.
    DIR *dr = opendir(".");

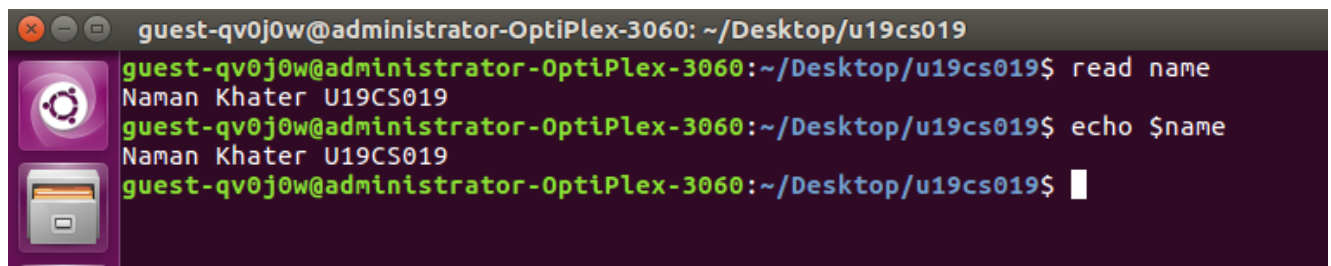
    if (dr == NULL) // opendir returns NULL if couldn't open directory
    {
        printf("Could not open current directory" );
        return 0;
    }
    while ((de = readdir(dr)) != NULL)
        printf("%s\n", de->d_name);

    closedir(dr);
    return 0;
}

guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ g++ opendir.c
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ ./a.out
.
..
a.out
opendir.c
wait.c
exit.c
getpid.c
sample
exec.c
sample.out
sample.c
fork.c
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$
```

READDIR

The readdir() function returns a pointer to a structure describing the next directory entry in the directory stream associated with dir.



```
guest-qv0j0w@administrator-OptiPlex-3060: ~/Desktop/u19cs019
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ read name
Naman Khater U19CS019
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ echo $name
Naman Khater U19CS019
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$
```

CHDIR

The chdir command is a system function (system call) which is used to change the current working directory.



```
guest-qv0j0w@administrator-OptiPlex-3060: ~/Desktop/u19cs019
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ cat chdir.c
#include<stdio.h>

// chdir function is declared
// inside this header
#include<unistd.h>
int main()
{
    char s[100];

    // printing current working directory
    printf("%s\n", getcwd(s, 100));

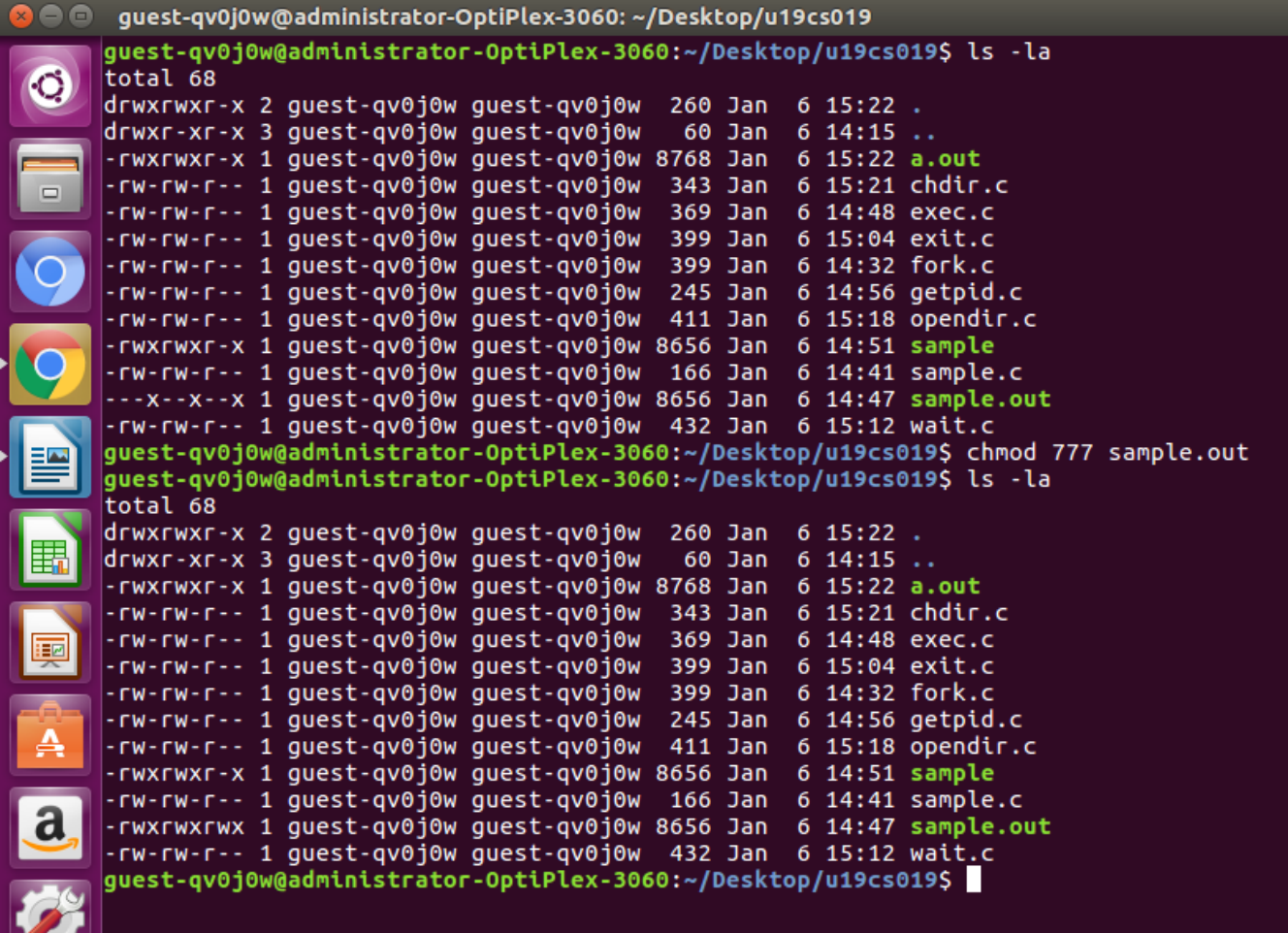
    // using the command
    chdir("../");

    // printing current working directory
    printf("%s\n", getcwd(s, 100));

    // after chdir is executed
    return 0;
}
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ g++ chdir.c
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ ./a.out
/tmp/guest-qv0j0w/Desktop/u19cs019
/tmp/guest-qv0j0w/Desktop
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$
```


CHMOD

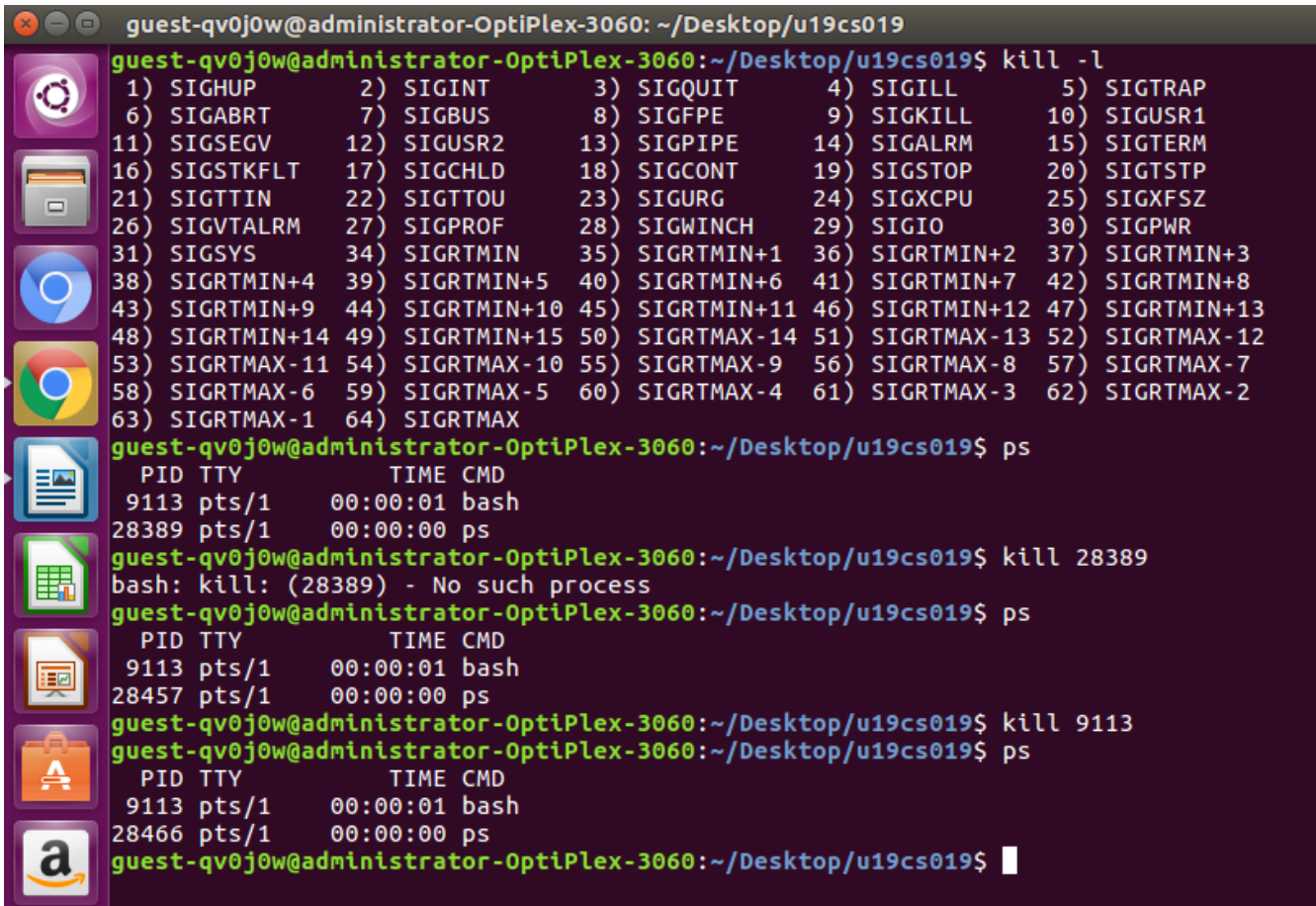
The chmod command is used to change the access mode of a file.



```
guest-qv0j0w@administrator-OptiPlex-3060: ~/Desktop/u19cs019
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ ls -la
total 68
drwxrwxr-x 2 guest-qv0j0w guest-qv0j0w 260 Jan 6 15:22 .
drwxr-xr-x 3 guest-qv0j0w guest-qv0j0w 60 Jan 6 14:15 ..
-rwxrwxr-x 1 guest-qv0j0w guest-qv0j0w 8768 Jan 6 15:22 a.out
-rw-rw-r-- 1 guest-qv0j0w guest-qv0j0w 343 Jan 6 15:21 chdir.c
-rw-rw-r-- 1 guest-qv0j0w guest-qv0j0w 369 Jan 6 14:48 exec.c
-rw-rw-r-- 1 guest-qv0j0w guest-qv0j0w 399 Jan 6 15:04 exit.c
-rw-rw-r-- 1 guest-qv0j0w guest-qv0j0w 399 Jan 6 14:32 fork.c
-rw-rw-r-- 1 guest-qv0j0w guest-qv0j0w 245 Jan 6 14:56 getpid.c
-rw-rw-r-- 1 guest-qv0j0w guest-qv0j0w 411 Jan 6 15:18 opendir.c
-rwxrwxr-x 1 guest-qv0j0w guest-qv0j0w 8656 Jan 6 14:51 sample
-rw-rw-r-- 1 guest-qv0j0w guest-qv0j0w 166 Jan 6 14:41 sample.c
---x--x--x 1 guest-qv0j0w guest-qv0j0w 8656 Jan 6 14:47 sample.out
-rw-rw-r-- 1 guest-qv0j0w guest-qv0j0w 432 Jan 6 15:12 wait.c
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ chmod 777 sample.out
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ ls -la
total 68
drwxrwxr-x 2 guest-qv0j0w guest-qv0j0w 260 Jan 6 15:22 .
drwxr-xr-x 3 guest-qv0j0w guest-qv0j0w 60 Jan 6 14:15 ..
-rwxrwxr-x 1 guest-qv0j0w guest-qv0j0w 8768 Jan 6 15:22 a.out
-rw-rw-r-- 1 guest-qv0j0w guest-qv0j0w 343 Jan 6 15:21 chdir.c
-rw-rw-r-- 1 guest-qv0j0w guest-qv0j0w 369 Jan 6 14:48 exec.c
-rw-rw-r-- 1 guest-qv0j0w guest-qv0j0w 399 Jan 6 15:04 exit.c
-rw-rw-r-- 1 guest-qv0j0w guest-qv0j0w 399 Jan 6 14:32 fork.c
-rw-rw-r-- 1 guest-qv0j0w guest-qv0j0w 245 Jan 6 14:56 getpid.c
-rw-rw-r-- 1 guest-qv0j0w guest-qv0j0w 411 Jan 6 15:18 opendir.c
-rwxrwxr-x 1 guest-qv0j0w guest-qv0j0w 8656 Jan 6 14:51 sample
-rw-rw-r-- 1 guest-qv0j0w guest-qv0j0w 166 Jan 6 14:41 sample.c
-rwxrwxrwx 1 guest-qv0j0w guest-qv0j0w 8656 Jan 6 14:47 sample.out
-rw-rw-r-- 1 guest-qv0j0w guest-qv0j0w 432 Jan 6 15:12 wait.c
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$
```

KILL

kill is a built-in command which is used to terminate processes manually.



```

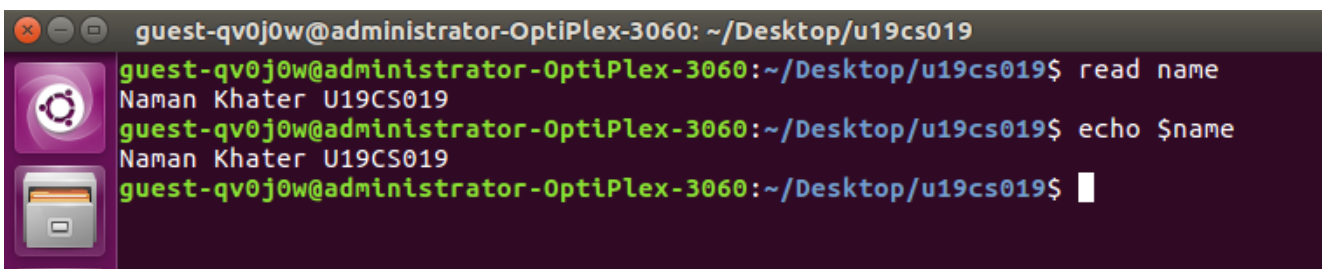
guest-qv0j0w@administrator-OptiPlex-3060: ~/Desktop/u19cs019
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
 6) SIGABRT     7) SIGBUS      8) SIGFPE      9) SIGKILL     10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM     15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD   18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN    22) SIGTTOU   23) SIGURG     24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF   28) SIGWINCH   29) SIGIO        30) SIGPWR
31) SIGSYS     34) SIGRTMIN  35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5 60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX

guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ ps
  PID TTY          TIME CMD
  9113 pts/1        00:00:01 bash
 28389 pts/1        00:00:00 ps
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ kill 28389
bash: kill: (28389) - No such process
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ ps
  PID TTY          TIME CMD
  9113 pts/1        00:00:01 bash
 28457 pts/1        00:00:00 ps
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ kill 9113
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ ps
  PID TTY          TIME CMD
  9113 pts/1        00:00:01 bash
 28466 pts/1        00:00:00 ps
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$

```

READ

From the file indicated by the file descriptor fd, the read() function reads cnt bytes of input into the memory area indicated by buf.



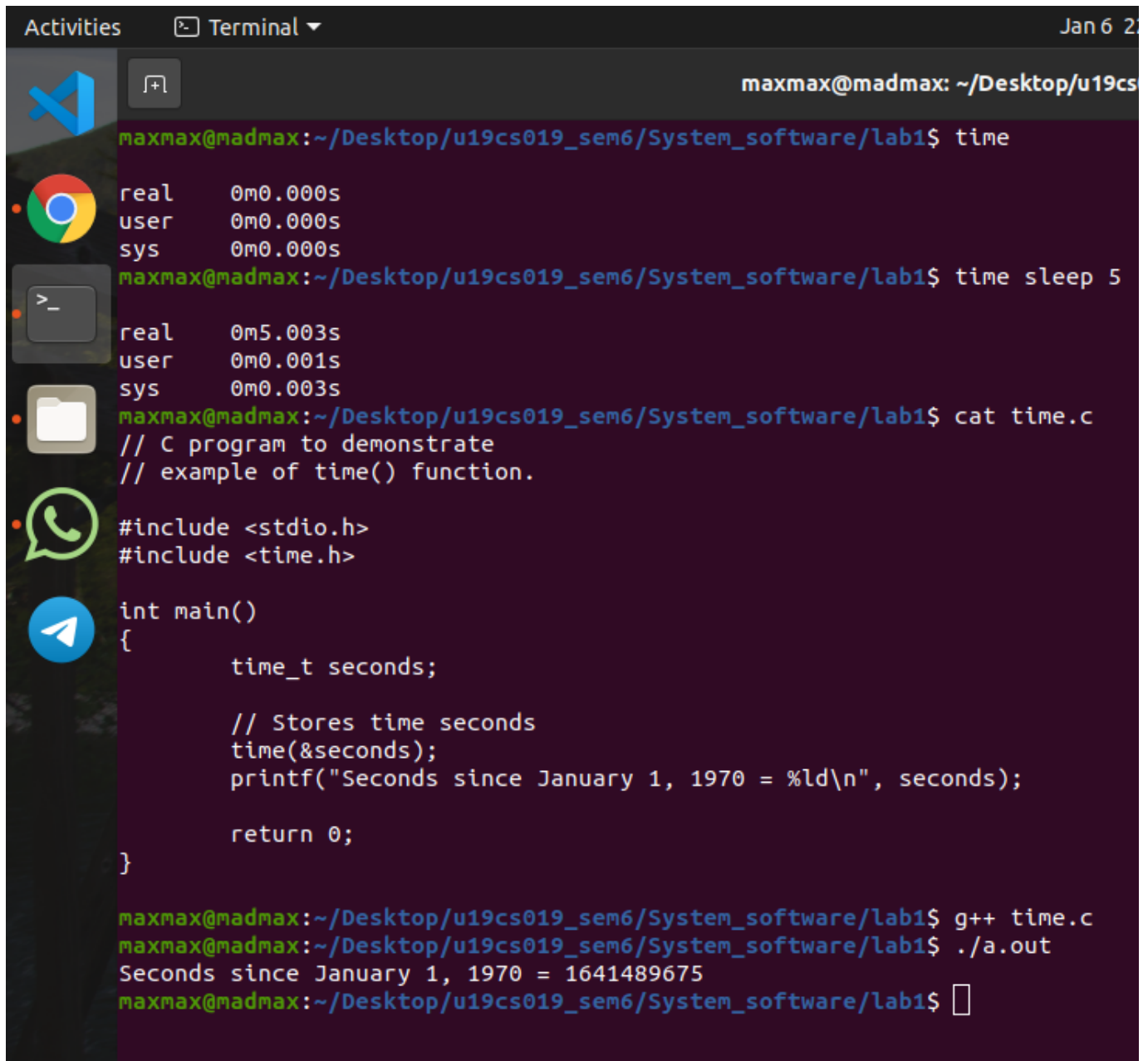
```

guest-qv0j0w@administrator-OptiPlex-3060: ~/Desktop/u19cs019
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ read name
Naman Khater U19CS019
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$ echo $name
Naman Khater U19CS019
guest-qv0j0w@administrator-OptiPlex-3060:~/Desktop/u19cs019$

```


TIME

time() returns the time since 00:00:00 UTC, January 1, 1970 (Unix timestamp) in seconds.



```
maxmax@madmax: ~/Desktop/u19cs019_sem6/System_software/lab1$ time
real    0m0.000s
user    0m0.000s
sys     0m0.000s
maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab1$ time sleep 5
real    0m5.003s
user    0m0.001s
sys     0m0.003s
maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab1$ cat time.c
// C program to demonstrate
// example of time() function.

#include <stdio.h>
#include <time.h>

int main()
{
    time_t seconds;

    // Stores time seconds
    time(&seconds);
    printf("Seconds since January 1, 1970 = %ld\n", seconds);

    return 0;
}

maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab1$ g++ time.c
maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab1$ ./a.out
Seconds since January 1, 1970 = 1641489675
maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab1$
```

OPEN && CLOSE

open() is used to open the file for reading, writing or both. close() tells the operating system you are done with a file descriptor and Close the file which pointed by fd.

```
maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab1$ cat open_close.c
// C program to illustrate close system Call
#include<stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

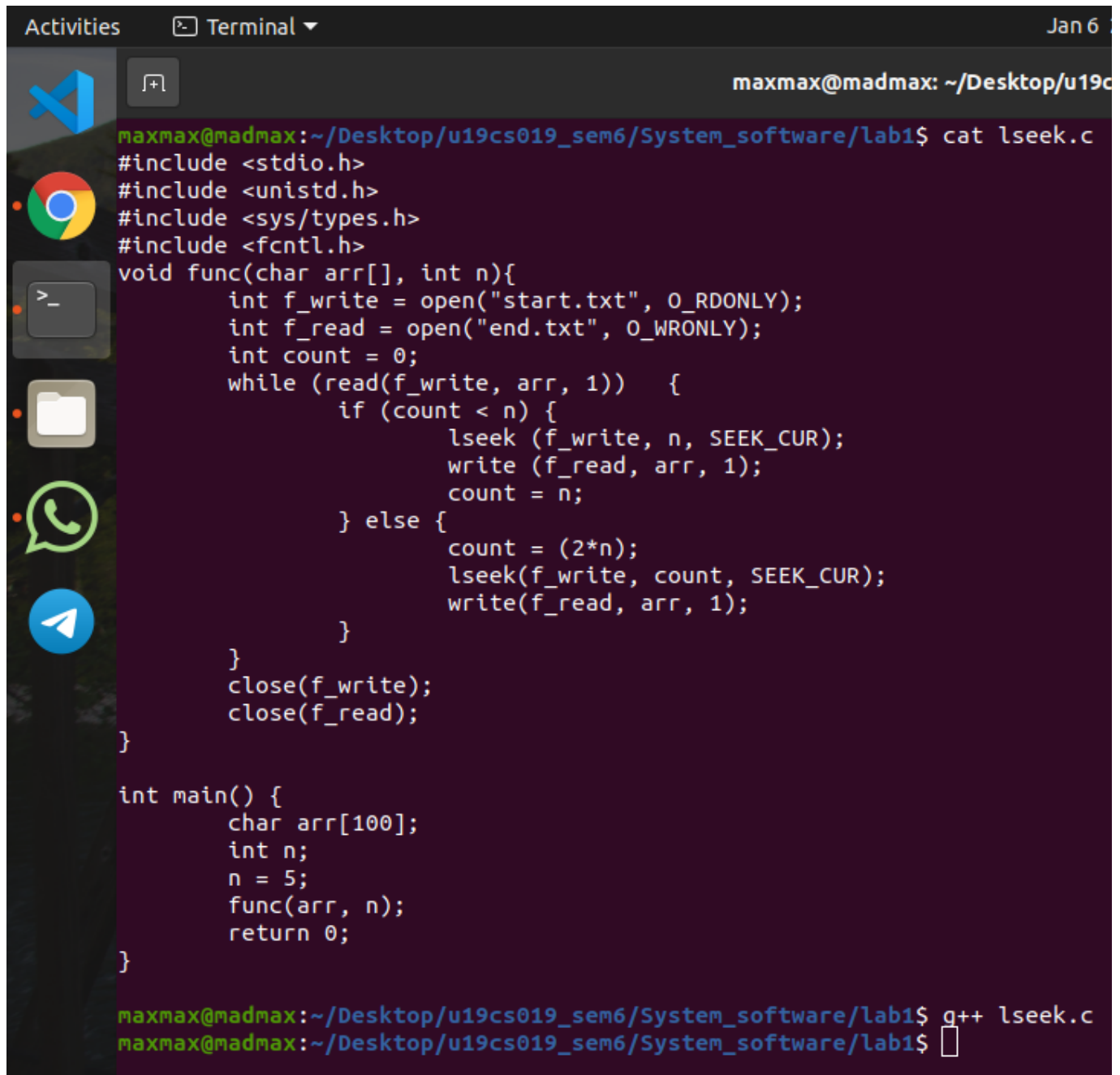
int main()
{
    int fd1 = open("sample.txt", O_RDONLY);
    if (fd1 < 0)
    {
        perror("c1");
        exit(1);
    }
    printf("opened the fd = % d\n", fd1);

    // Using close system Call
    if (close(fd1) < 0)
    {
        perror("c1");
        exit(1);
    }
    printf("closed the fd.\n");
}

maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab1$ g++ open_close.c
maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab1$ ./a.out
opened the fd = 3
closed the fd.
maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab1$
```

LSEEK

lseek() repositions the file offset of the open file description associated with the file descriptor fd to the argument offset according to the directive whence as follows



```
maxmax@madmax: ~/Desktop/u19c
maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab1$ cat lseek.c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>
void func(char arr[], int n){
    int f_write = open("start.txt", O_RDONLY);
    int f_read = open("end.txt", O_WRONLY);
    int count = 0;
    while (read(f_write, arr, 1)) {
        if (count < n) {
            lseek (f_write, n, SEEK_CUR);
            write (f_read, arr, 1);
            count = n;
        } else {
            count = (2*n);
            lseek(f_write, count, SEEK_CUR);
            write(f_read, arr, 1);
        }
    }
    close(f_write);
    close(f_read);
}

int main() {
    char arr[100];
    int n;
    n = 5;
    func(arr, n);
    return 0;
}

maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab1$ g++ lseek.c
maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab1$
```

```
maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab1$ cat start.txt
Lorem ipsum dolor sit amet consectetur adipisicing elit. Maxime mollitia,
molestiae quas vel sint commodi repudiandae consequuntur voluptatum laborum
numquam blanditiis harum quisquam eius sed odit fugiat iusto fuga praesentium
optio, eaque rerum! Provident similique accusantium nemo autem. Veritatis
obcaecati tenetur iure eius earum ut molestias architecto voluptate aliquam
nihil, eveniet aliquid culpa officia aut! Impedit sit sunt quaerat, odit,
tenetur error, harum nesciunt ipsum debitis quas aliquid. Reprehenderit,
quia. Quo neque error repudiandae fuga? Ipsa laudantium molestias eos
sapiente officiis modi at sunt excepturi expedita sint? Sed quibusdam
recusandae alias error harum maxime adipisci amet laborum. Perspiciatis
minima nesciunt dolorem! Officiis iure rerum voluptates a cumque velit
quibusdam sed amet tempora. Sit laborum ab, eius fugit doloribus tenetur
fugiat, temporibus enim commodi iusto libero magni deleniti quod quam
consequuntur! Commodi minima excepturi repudiandae velit hic maxime
doloremque. Quaerat provident commodi consectetur veniam similique ad
earum omnis ipsum saepe, voluptas, hic voluptates pariatur est explicabo
fugiat, dolorum eligendi quam cupiditate excepturi mollitia maiores labore
suscipit quas? Nulla, placeat. Voluptatem quaerat non architecto ab laudantium
modi minima sunt esse temporibus sint culpa, recusandae aliquam numquam
totam ratione voluptas quod exercitationem fuga. Possimus quis earum veniam
quasi aliquam eligendi, placeat qui corporis!
maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab1$ cat end.txt
maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab1$ ./a.out
maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab1$ cat end.txt
Li oa m, i ett
armo mQpiceVbneuso htuaadtouanbap, rasuseisua ra m uim feuu e ,t uo smiauoeevaernc ie e,ttlimqtuaaiNeattudumneluruenosiagepmaxmax@madmax:~/De
sktop/u19cs019_sem6/System_software/lab1$ □
```

MOUNT

mount() attaches the filesystem specified by source (which is often a pathname referring to a device, but can also be the pathname of a directory or file, or a dummy string) to the location (a directory or file) specified by the pathname in target.

CHOWN

chown command is used to change the file owner or group.