**Operating System**

**Assignment 01 Roll No.: CE092**

**Aim:** Implementation of “cat” and “cp” command in C. (use of open, read, write, and close system calls)

**What are the System Calls?**

In computing, a **system call** is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to **interact with the operating system**. A computer program makes a system call when it makes a request to the operating system’s kernel. System call **provides** the services of the operating system to the user programs via Application Program Interface (API).

**What are the file descriptors?**

In simple words, when you open a file, the operating system creates an entry to represent that file and store the information about that opened file. So, if there are 100 files opened in your OS then there will be 100 entries in OS (somewhere in kernel). These entries are represented by integers like (...100, 101, 102....). This entry number is the file descriptor. So, it is just an integer number that uniquely represents an opened file in operating system. If your process opens 10 files then your Process table will have 10 entries for file descriptors.

|  |  |  |  |
| --- | --- | --- | --- |
| **Descriptive Name** | **Short Name** | **File Number** | **Description** |
| Standard In | Stdin | 0 | Input from the keyboard |
| Standard Out | Stdout | 1 | Output to the console |
| Standard Error | stderr | 2 | Error output to the console |

**Various System Calls**

**Read: read from a file descriptor**

**Synopsis:** #include<unistd.h>

**ssize\_t read (int** *fd***, void \****buf***, size\_t** *count***);**

**read** () attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*.

On success, the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number.

On error, -1 is returned, and *errno* is set appropriately.

Example: n = read (0, buff, sizeof(buff))

**Write: write to a file descriptor**

**Synopsis: #**include<unistd.h>

**ssize\_t write (int** *fd***, const void \****buf***, size\_t** *count***);**

**write** () writes up to *count* bytes from the buffer pointed *buf* to the file referred to by the file descriptor *fd*.

On success, the number of bytes written is returned (zero indicates nothing was written). On error, -1 is returned, and *errno* is set appropriately.

Example: write (1 , buff , n)

**Open: open and possibly create a file**

**Synopsis:** #include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

I int open(const char \*pathname, int flags);

int open(const char \*pathname, int flags, mode\_t mode);

**write** () writes up to *count* bytes from the buffer pointed *buf* to the file referred to by the file descriptor *fd*.

On success, the number of bytes written is returned (zero indicates nothing was written). On error, -1 is returned, and *errno* is set appropriately.

Example: write (1 , buff , n)

**Close: close a file descriptor**

**Synopsis:** #include <unistd.h>

int close(int *fd*);

**close**() closes a file descriptor, so that it no longer refers to any file and may be reused.

**close**() returns zero on success. On error, -1 is returned, and *errno* is set appropriately.

Example: close(fd)

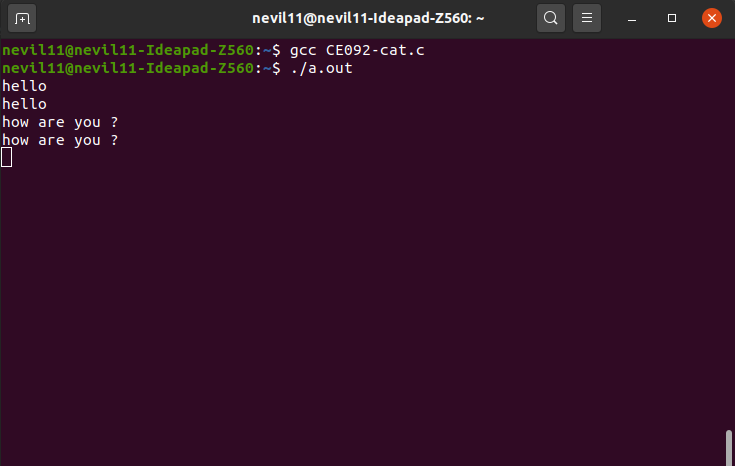
**Tasks**

1. Implement basic “cat” command using system calls.

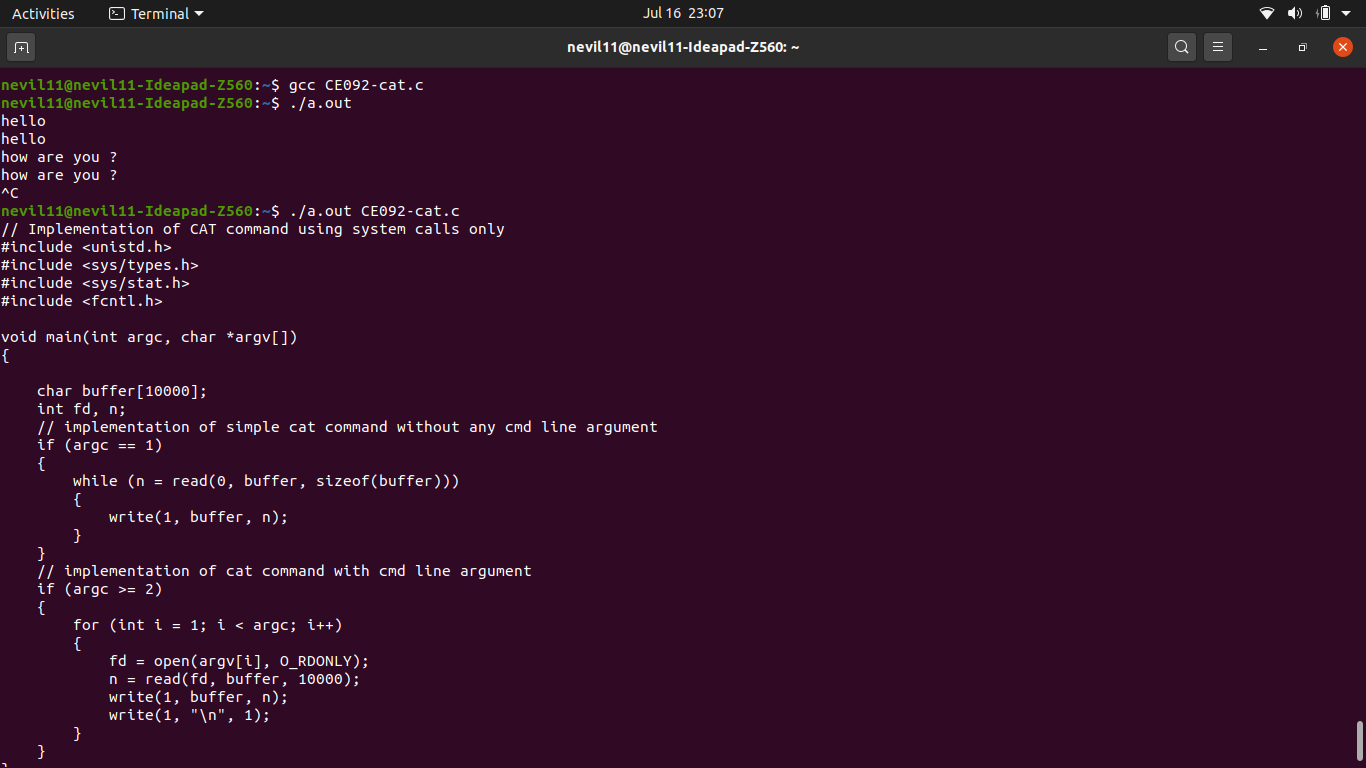
* Code



* Output :



Normal cat behaviour of cat command without command line arguments



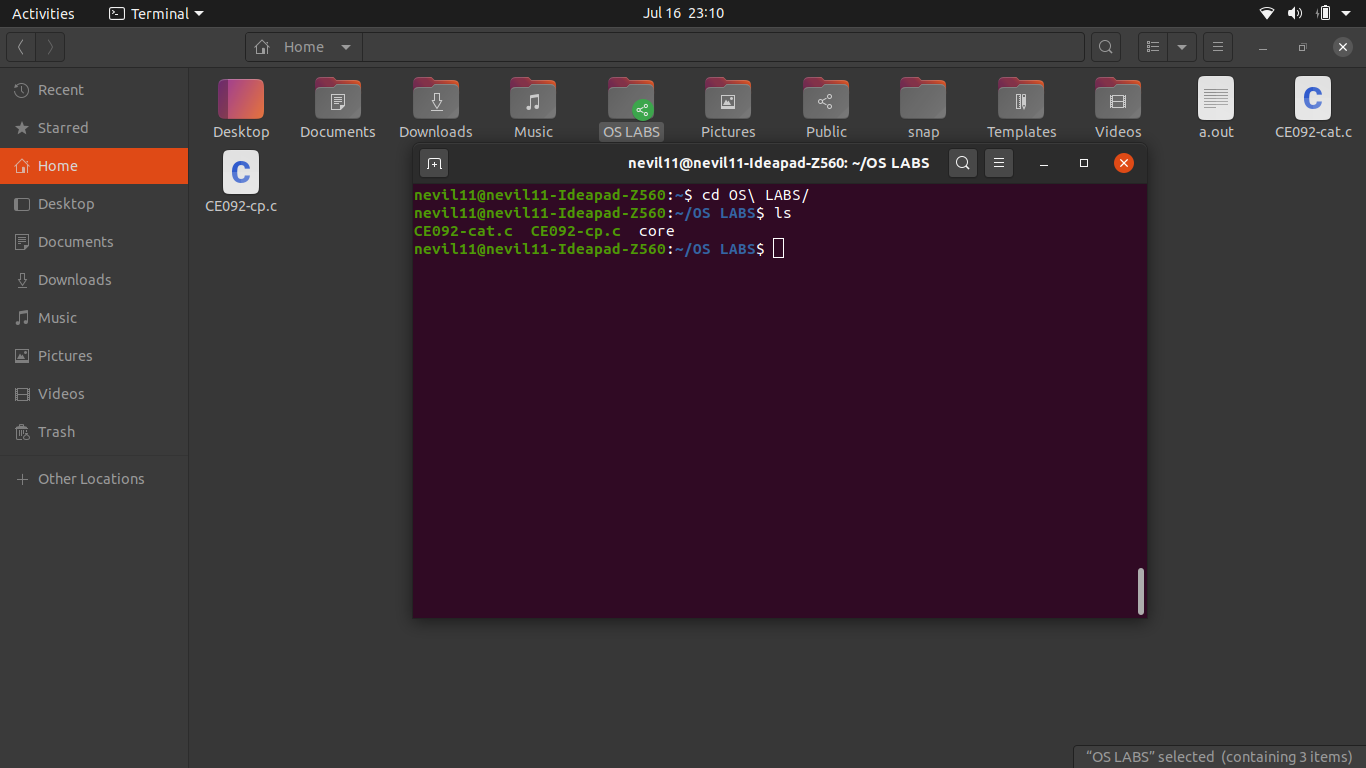
Cat command behaviour with command line arguments.

1. Implement basic “cp” command using system calls

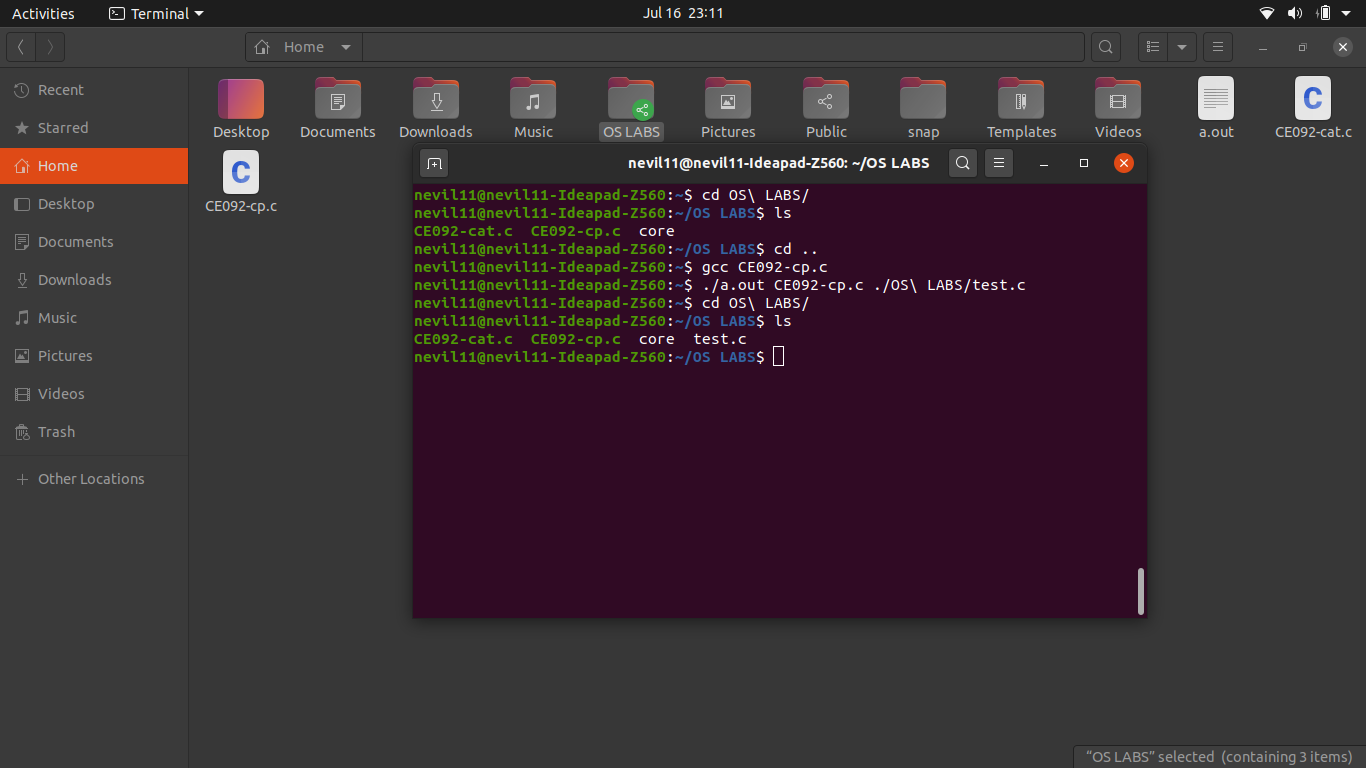
* Code



* Output



Before running our a.out file to copy CE092-cp.c as test.c inside OS LABS folder.



After running a.out file with command line arguments, we can see the file test.c has been created inside OS LABS folder.