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))

<u>Write:</u> 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)

<u>Open:</u> 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)

<u>Close:</u> 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)

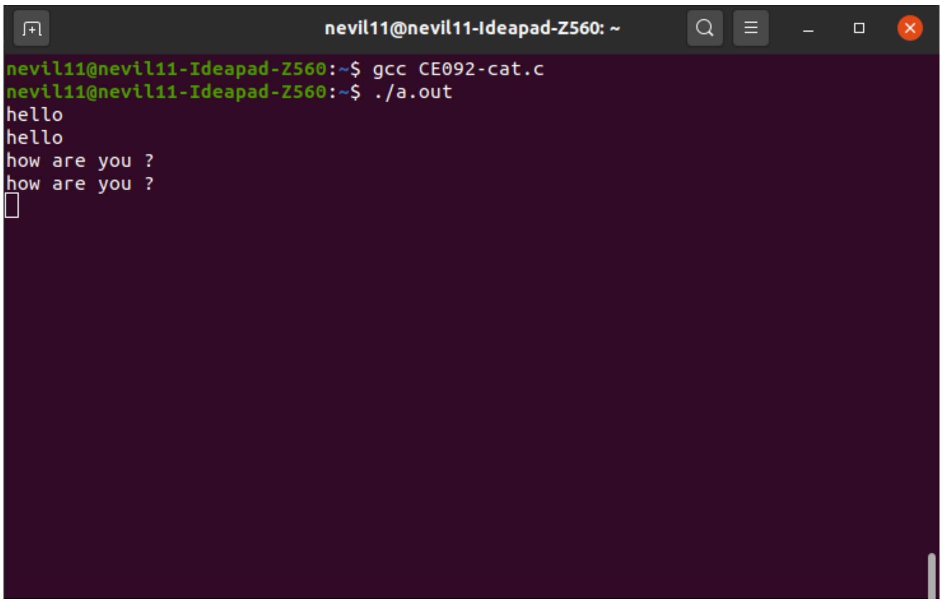
Tasks

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

- Code

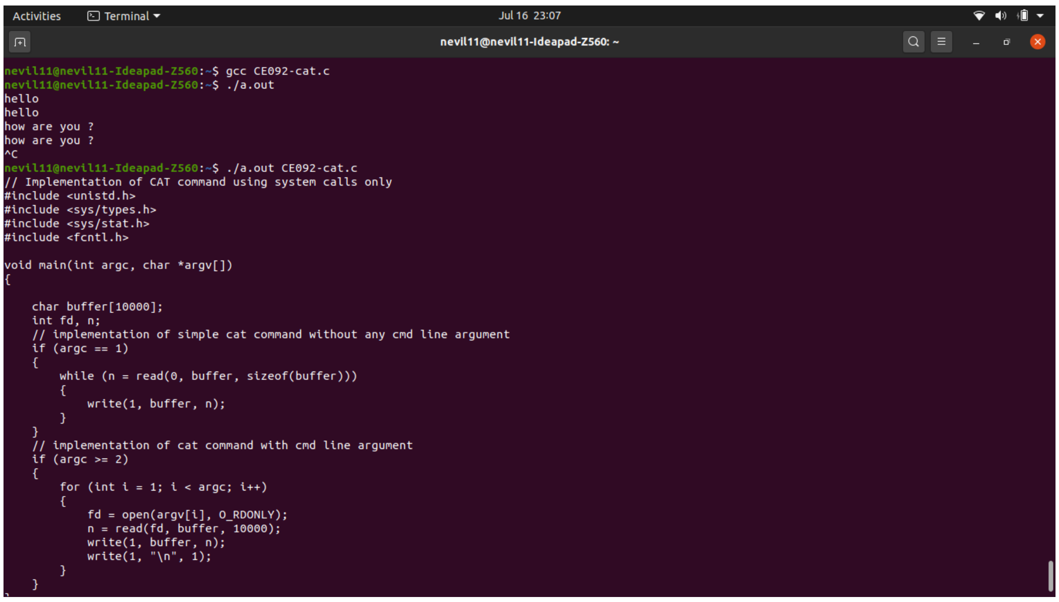
```c
// Implementation of CAT command using system calls only
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

void main(int argc, char *argv[])
{

    char buffer[10000];
    int fd, n;
    // implementation of simple cat command without any cmd
line argument
    if (argc == 1)
    {
        while (n = read(0, buffer, sizeof(buffer)))
        {
            write(1, buffer, n);
        }
    }
    // implementation of cat command with cmd line argument
    if (argc >= 2)
    {
        for (int i = 1; i < argc; i++)
        {
            fd = open(argv[i], O_RDONLY);
            n = read(fd, buffer, 10000);
            write(1, buffer, n);
            write(1, "\n", 1);
        }
    }
}
```

- Output :



```
nevil11@nevil11-Ideapad-Z560:~$ gcc CE092-cat.c
nevil11@nevil11-Ideapad-Z560:~$ ./a.out
hello
hello
how are you ?
how are you ?
```

Normal cat behaviour of cat command without command line arguments



```
nevil11@nevil11-Ideapad-Z560:~$ gcc CE092-cat.c
nevil11@nevil11-Ideapad-Z560:~$ ./a.out
hello
hello
how are you ?
how are you ?
^C
nevil11@nevil11-Ideapad-Z560:~$ ./a.out CE092-cat.c
// Implementation of CAT command using system calls only
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

void main(int argc, char *argv[])
{

    char buffer[10000];
    int fd, n;
    // implementation of simple cat command without any cmd line argument
    if (argc == 1)
    {
        while (n = read(0, buffer, sizeof(buffer)))
        {
            write(1, buffer, n);
        }
    }
    // implementation of cat command with cmd line argument
    if (argc >= 2)
    {
        for (int i = 1; i < argc; i++)
        {
            fd = open(argv[i], O_RDONLY);
            n = read(fd, buffer, 10000);
            write(1, buffer, n);
            write(1, "\n", 1);
        }
    }
```

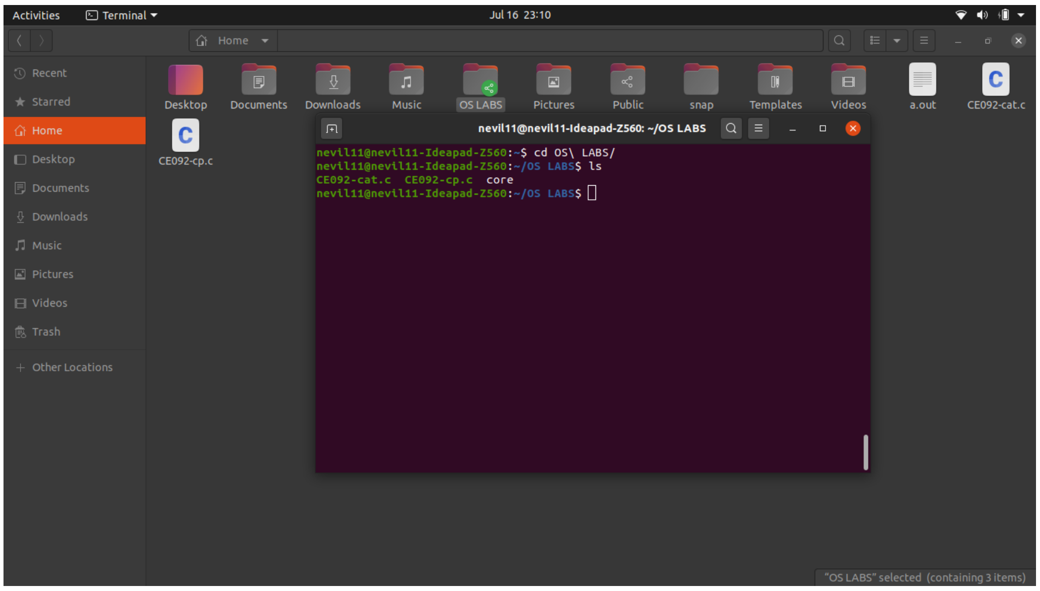
Cat command behaviour with command line arguments.

2. Implement basic “cp” command using system calls

- Code

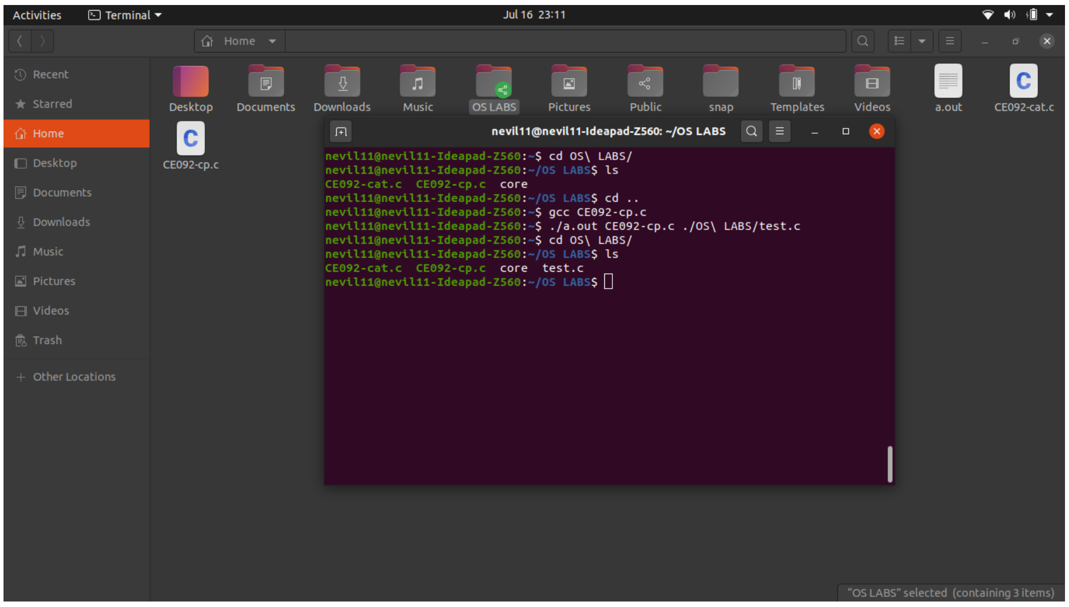
```c
// Implementation of basic cp command using system calls
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/types.h>
#include <stdio.h>

int main(int argc, char *argv[]){
    int n;
    char buff[100000];
    if(argc==1 || argc==2)
    {
        write(1,"too less argument\n",18);
    }
    else{
        int fdsrc,fddest;
        fdsrc=open(argv[1],O_RDONLY);
        if(fdsrc==2)
        {
            write(1,"source file doesn't exists\n",27);
            return 0;
        }
        fddest=open(argv[2],O_RDWR|O_CREAT);
        n=read(fdsrc,buff,sizeof(buff));
        write(fddest,buff,n);
        close(fddest);
        close(fdsrc);
    }
    return 0;
}
```

- Output



```
nevil11@nevil11-Ideapad-Z560: ~/OS LABS
nevil11@nevil11-Ideapad-Z560:~$ cd OS\ LABS/
nevil11@nevil11-Ideapad-Z560:~/OS LABS$ ls
CE092-cat.c  CE092-cp.c  core
nevil11@nevil11-Ideapad-Z560:~/OS LABS$ 
```

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



```
nevil11@nevil11-Ideapad-Z560: ~/OS LABS
nevil11@nevil11-Ideapad-Z560:~$ cd OS\ LABS/
nevil11@nevil11-Ideapad-Z560:~/OS LABS$ ls
CE092-cat.c  CE092-cp.c  core
nevil11@nevil11-Ideapad-Z560:~/OS LABS$ cd ..
nevil11@nevil11-Ideapad-Z560:~$ gcc CE092-cp.c
nevil11@nevil11-Ideapad-Z560:~$ ./a.out CE092-cp.c ./OS\ LABS/test.c
nevil11@nevil11-Ideapad-Z560:~$ cd OS\ LABS/
nevil11@nevil11-Ideapad-Z560:~/OS LABS$ ls
CE092-cat.c  CE092-cp.c  core  test.c
nevil11@nevil11-Ideapad-Z560:~/OS LABS$ 
```

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