Assignment 2
File/Directory Operations System Calls on Linux

Submission Date: 16th August 2015, 3:00 pm

Goal:

This Assignment based on common system calls for making input output operations on files and also for handling files and directories in the Linux operating system. First, I have described some background information on the relevant systems calls. Then, I have given you three tasks.

System Calls for File Operations

The common system calls for file descriptors: *create*, *open*, *lseek* and so on.

OPEN

Opening or creating a file can be done using the system call open. The syntax is:

```
int open(const char *path, int flags, ... /* mode_t mod */);
```

CREAT

A new file can be created by:

```
int creat(const char *path, mode_t mod);
```

READ

When we want to read a certain number of bytes starting from the current position in a file, we use the *read* call. The syntax is:

```
ssize_t read(int fd, void* buf, size_t noct);
```

WRITE

For writing a certain number of bytes into a file starting from the current position we use the *write* call. Its syntax is:

```
ssize_t write(int fd, const void* buf, size_t noct);
```

CLOSE

For closing a file and thus eliminating the assigned descriptor we use the system call *close*.

```
int close(int fd);
```

LSEEK

To position a pointer (that points to the current position) in an absolute or relative way can be done by calling the *lseek* function. Read and write operations are done relative to the current position in the file. The syntax for *lseek* is:

```
off_t lseek(int fd, off_t offset, int ref);
```

STAT, LSTAT and FSTAT

In order to obtain more details about a file the following system calls can be used: *stat*, *lstat* or *fstat*.

```
int stat(const char* path, struct stat* buf);
int lstat(const char* path, struct stat* buf);
int fstat(int df, struct stat* buf);
```

Functions for working with directories

A directory can be read as a file by anyone whoever has reading permissions for it. Writing a directory as a file can only be done by the kernel. The structure of the directory appears to the user

as a succession of structures named directory entries. A directory entry contains, among other information, the name of the file and the i-node of this. For reading the directory entries one after the other we can use the following functions:

```
#include <sys/types.h>
#include <dirent.h>
DIR* opendir(const char* pathname);
struct dirent* readdir(DIR* dp);
void rewinddir(DIR* dp);
int closedir(DIR* dp);
```

The *opendir* function opens a directory. It returns a valid pointer if the opening was successful and NULL otherwise.

The *readdir* function, at every call, reads another directory entry from the current directory. The first readdir will read the first directory entry; the second call will read the next entry and so on. In case of a successful reading the function will return a valid pointer to a structure of type dirent and NULL otherwise (in case it reached the end of the directory, for example)

The *rewinddir* function repositions the file pointer to the first directory entry (the beginning of the directory).

The *closedir* function closes a previously opened directory. In case of an error it returns the value -1.

Tasks:

You have to perform the following three tasks as a part of this assignment.

1. Write a program that writes the lines of a file into another file, but in the reverse order. The names of the files should be read as input parameters. Ex-

Input.txt:
There are three statements in this file.
This is statement 1.
This is statement 2.

Output.txt:
This is statement 2.
This is statement 1.
There are three statements in this file.

2. Calculate the size of a file using the ***lseek*** command. Once you found out the size, calculate the number of blocks assigned for the file. Compare these results with the similar results obtained when using the function ***stat***.

3. Write a program that deletes a directory with all its subdirectories and files.

Input & Output

The input to the assignment for task 1 is the file: t1-inp.txt. The corresponding output for task1 will be t1-out.txt. The contents of these files are as described above.

For task2, input is a file, t2-inp.txt. The output will be displayed on the terminal as follow:
lseek size:
stat size:

For task3, the input & output is given at the command prompt as follows:
$ ./task3
successfully deleted
$
If your program is not able to delete the directory for some reason, then it should display the corresponding error message.

Submission Instructions:

Develop all the three tasks in C language. Name your files as task1.c, task1.c, task3.c. Then zip all the files and name the zipped archive as ***assgn2-<rollno>.zip***. Submit your assignment by 16th August 2015, 3:00 pm.

Reading Materials and other Useful Links:

1.http://os.obs.utcluj.ro/OS/Lab/03.System%20Calls%20for%20File%20and%20Directory%20Manipulation%20in%20Linux.html
Most of the text in this assignment has been borrowed from this link.

2.http://man7.org/linux/man-pages/man2/syscalls.2.html
3.http://www.digilife.be/quickreferences/qrc/linux%20system%20call%20quick%20reference.pdf
4.http://www.tldp.org/HOWTO/html_single/Implement-Sys-Call-Linux-2.6-i386/
5.http://reiber.org/nxt/pub/Linux/LinuxKernelDevelopment/Linux.Kernel.Development.3rd.Edition.pdf