CS-362L Operating System Lab 03

Assignment 3

Linux system calls related to file system call, I/O system call.

Objectives: To get familiar with different system calls used for file and I/O device management.

Processing steps:

What is System Call?

A system call is just what its name implies—a request for the operating system to do something on behalf of the user's program.

File descriptors

File descriptor is integer that uniquely identifies an open file of the process.

Step 1: Study create() system call.

Used to Create a new empty file. The syntax is:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int creat(const char *path, mode_t mod);
```

Parameter

- filename : name of the file which you want to create.
- mode : indicates permissions of new file.

Return Value:

- return first unused file descriptor (generally 3 when first create use in process because 0, 1, 2 fd are reserved.
- return -1 when error.

Step 2: Study open() system call.

Open() lets you open a file for reading, writing, or reading and writing. The syntax is:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *path,
         int flags,... /* mode_t mod */);
```

This function returns the file descriptor or in case of an error -1. The number of arguments that this function can have is two or three. The third argument is used only when creating a new file. When we want to open an existing file only

two arguments are used. The common flags used are:

O_RDONLY
Opens the file for reading.
O_WRONLY
Opens the file for writing.
O_RDWR
The file is opened for reading and writing.
O_APPEND
It writes successively to the end of the file.
O_CREAT
The file is created in case it didn\92t already exist.

Parameter

- Path : path to file which you want to use.
- flags : how you like to use.

Return Value:

- Return file descriptor used, -1 upon failure.

Step 3: Study read() system call.
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:

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

The function returns the number of bytes read, 0 for end of file (EOF) and -1 in case an error occurred. It reads cnt bytes from the open file referred by the fd descriptor and it puts it into a buffer buf.
Parameter

- fd: file descripter.
- buf: buffer to read data from.
- cnt: length of buffer.

Return Value:

- return Number of bytes read on success.
- return -1 on error.

Step 4: Study write() system call.
For writing a certain number of bytes into a file starting from the current position we use the write call. Its syntax is:

```
#include <fcntl.h>
size_t write (int fd, void* buf, size_t cnt);
```

Parameter

- fd: file descripter.
- buf: buffer to write data to.
- cnt: length of buffer.

Return Value:

- return Number of bytes written on success.
- return -1 on error.

Step 5: Study Close() system call

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

```
#include <unistd.h>
int close(int fd);
```

The function returns 0 in case of success and -1 in case of an error.

Step 6: Working with directories

A directory can be read as a file by anyone whoever has reading permissions for it.

Creating / Removing Directory: Directories can be created and removed using mkdir and rmdir function calls. The function prototypes are:

int mkdir(char* name, int mode);

int rmdir(char* name);

returns 0 or 1 for success or failure. For example, creating a new directory called "newfiles" with only the READ access for the user we can do the following.

mkdir("newfiles", 0400);

Later you can remove this directory by calling

rmdir("newfiles");

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 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.

Conclusion: At the end of this lab, student will be able to use the most common system calls in order to make input-output operations on files, as well as operations to handle files and directories in the Linux operating system.

Class Activity

1. Write a C program to implement the above mentioned file system calls.

   To write a 'c' program for I/O system calls.

   ALGORITHM:

   1. Start the programs

   2. open a file for O_RDWR for R/W,O_CREATE for creating a file , O_TRUNC for truncate a file

   3. Using getchar(), read the character and stored in the string[] array

   4. The string [] array is write into a file close it.

   5. Then the first is opened for read only mode and read the characters and displayed It and close the file

   6. Stop the program

   Write a program to take id name and CGPA and write in a file using IO system calls. Your program also read id name and CGPA from file and print on console.

```
#include <fcntl.h>
#include <stdio.h>
#include <zconf.h>
main( )
{
 char id[20], name[50],CGPA[5];
 char Rid[20], Rname[50],RCGPA[4];
 int fp = open("file",O_RDWR|O_CREAT);
```

```
if(fp != -1)
{
printf("Enter ID : ");
fgets(id, sizeof(id),stdin);
printf("Enter Name : ");
fgets(name, sizeof(name),stdin);
printf("Enter CGPA : ");
fgets(CGPA, sizeof(CGPA),stdin);
printf("All records write using write() System Calls\n");
write(fp,id, sizeof(id));
write(fp,name, sizeof(name));
write(fp,CGPA, sizeof(CGPA));
lseek(fp,0,0);
printf("All records Read using Read() System Calls\n");
read(fp,Rid, sizeof(id));
read(fp,Rname, sizeof(name));
read(fp,RCGPA, sizeof(CGPA));
printf("ID : %s\nName : %s\nCGPA : %s\n",Rid,Rname,RCGPA);
close(fp);
}
else
{
printf("Ops Error");
}
return
```

output

Enter ID : Your ID

Enter Name : Your Name

Enter CGPA : 3.72

All records write using write() System Calls

All records Read using Read() System Calls

ID : Your ID

Name : Your Name

CGPA : 3.72

Process finished with exit code 0

Exercise

1. Write a program that copies the contents of an existing file into another file. The names of the two file should be read as an input from the command line.
2. Write a program that displays the contents of a directory, specifying the type for each of its files. The name for the directory should be an input parameter.
3. Write a program that will categorize all files in the current folder based on their file type. That is all .txt file in one folder called txt, all .bmp files in another folder called bmp etc. The argument to the program is a folder name.

References:

- ”System Call”
  http://man7.org/linux/man-pages/man2/syscalls.2.html
- ”System Call”
  https://www.cs.cmu.edu/~guna/15-123S11/Lectures/Lecture24.pdf