Programming Operations on Files and Directories in UNIX Lab. #5

Arif A. Balik

Undergraduate Student
Sytstems Programming
Department of Computer Science
Arel University
Büyükçekmece, İstanbul 34537
Email: arifbalik@outlook.com

May 23, 2019

Abstract

This reports covers some properties of system calls which relate to directory and file operations on UNIX environment. It explains its theory of operation and answers some questions asked in the lab sheet. This report is powered by LaTeX

Theory of Operation

The programs waits for a command line argument to work, that is path to be examined, if not, program returns an error and exits. After, programs immidiatly gathers information about the path, and checks whether path is a directory or file. Then if it is a file than program prints its information and exits, if it is a directory than it tries to print every file and directory in that directory along with their sizes, finally it closes file and exits.

Q&A

Q1. *What is the purpose of this lab work?*

The purpose of this laboratory work is to understand basic file system operations such as reading names and sizes of files and directories.

Q2. *What is the purpose of the system call stat()? What arguments are used in this system call?*

stat() system call is used to gather information about the file or directory.
The prototype of this function is as follows;

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

The first argument is path (directory or file) which function will gather information for, and the second one is the structure that function will fill the information with.

Q3. *What are main fields in the structure stat? In what header file is this structure defined?*

Based on *The Open Group Base Specifications Issue 6* structure is as follows;

```
dev_t     st_dev     ID of device containing file
ino_t     st_ino     file serial number
mode_t    st_mode    mode of file (see below)
nlink_t   st_nlink   number of links to the file
uid_t     st_uid     user ID of file
gid_t     st_gid     group ID of file
dev_t     st_rdev    device ID
                        (if file is character or block special)
off_t     st_size    file size in bytes
                        (if file is a regular file)
time_t    st_atime   time of last access
time_t    st_mtime   time of last data modification
time_t    st_ctime   time of last status change
blksize_t st_blksize a filesystem-specific preferred I/O block
                size for this object.  In some filesystem types,
                this may vary from file to file
blkcnt_t  st_blocks  number of blocks allocated for this object
```

it contains various information about file or directory such as size, time of last access, mode of file etc.
This structure can be accessed through *sys/stat.h*

Q4. *How can you learn that an item is a regular file or a directory?*

After calling stat system call, developer can mask the *st_mode* and *S_IFMT* with *&* operator and after this mask operation developer can check various options. All options are listed below;

```
S_IFBLK
Block special.
S_IFCHR
Character special.
S_IFIFO
FIFO special.
S_IFREG
Regular.
S_IFDIR
Directory.
S_IFLNK
Symbolic link.
S_IFSOCK
Socket.
```

Q5. *What is the purpose of the constants S_IFREG and S_IFDIR? In which header file these constants are defined?*

Those constant are *regular file* and *direction* respectively. It is used to determine the type of the path.

Those constants can be accessed through *sys/stat.h*

Q6 *What function can be used for opening a directory? What is the returned result of this function when the function call fails?*

opendir can be used to open a directory. When function fails it returns *NULL*[i] and sets the *errno*.

Q7. *How can you learn the size of a file in a directory?*

Size and many other features of a file or directory can be learned by using structure *stat*. For size information developer can look at *st_size* in *stat*.

[i] In Linux/UNIX a system call returning *NULL* or a positive number when something goes wrong is too rare.

Optimization

As asked in the lab sheet, owner of the file and searching all the sub directories will be implemented.

Owner of the file can be found in structure *stat*, the field is *st_uid*. Following code prints the information of the users for a given path.

```
struct passwd *ps = getpwuid(info.st_uid);
 printf("Owner: %s\n", ps->pw_name);
```

And following function can recursively searches for sub directories;

```
void list_dir(char *base_path, const int root)
{
    int i;
    char path[1000];
    struct dirent *dp;
    struct stat info;
    DIR *dir = opendir(base_path);
    char cur_path[BUFSIZ + 1];
    struct passwd *ps;

    if (!dir){
        return;
    }


    while ((dp = readdir(dir)) != NULL){

        if (strcmp(dp->d_name, ".") != 0 &&
                        strcmp(dp->d_name, "..") != 0){
            for (i=0; i<root; i++) {
                if (i % 2 == 0 || i == 0)
                    printf("|");
                else
                    printf(" ");

            }
        sprintf(cur_path, "%s/%s",base_path, dp->d_name);
            if(stat(cur_path, &info) < 0) return;
        ps = getpwuid(info.st_uid);
            printf("|--%s, Size: %ld, Owner: %s\n",
                dp->d_name, info.st_size, ps->pw_name) ;

            strcpy(path, base_path);
            strcat(path, "/");
            strcat(path, dp->d_name);
            list_dir(path, root + 2);
        usleep(100000); // to make it look coooool!
        }
    }

    closedir(dir);
}
```

Full source code is given in the next section.

Appendix

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/dir.h>
#include <string.h>
#include <pwd.h>


void  list_dir(char *base_path, const int root);

int main(int argc, char **argv)
{


    DIR *dp;
    struct dirent *dir;
    struct stat info;
    char pathname[BUFSIZ+1];
    struct passwd *ps;

    if(argc != 2){
        printf("Please give a path. \n
                        Example: ./[name] [path]\n");
        exit(1);
    }

    if(stat(argv[1], &info) < 0) exit(1);

    if((info.st_mode & __S_IFMT) != __S_IFREG &&
        (info.st_mode & __S_IFMT) != __S_IFDIR ){
        printf("%s is not a directory or file\n", argv[1]);
        exit(1);
    }

    switch((info.st_mode & __S_IFMT)){
        case __S_IFREG:
            ps = getpwuid(info.st_uid);
            printf("File name : %s, Size : %ld, Owner: %s\n",
                 argv[1], info.st_size, ps->pw_name);
            exit(1);
        break;
        case __S_IFDIR:

             list_dir(argv[1], 0);
        break;
    }

    closedir(dp);

    return 0;
}

void list_dir(char *base_path, const int root)
{
    int i;
    char path[1000];
    struct dirent *dp;
    struct stat info;
    DIR *dir = opendir(base_path);
    char cur_path[BUFSIZ + 1];
    struct passwd *ps;

    if (!dir){
        return;
    }


    while ((dp = readdir(dir)) != NULL){

        if (strcmp(dp->d_name, ".") != 0 && strcmp(dp->d_name, "..") != 0){
            for (i=0; i<root; i++) {
                if (i % 2 == 0 || i == 0)
                    printf("|");
                else
                    printf(" ");

            }
        sprintf(cur_path, "%s/%s",base_path, dp->d_name);
            if(stat(cur_path, &info) < 0) return;
        ps = getpwuid(info.st_uid);
            printf("|--%s, Size: %ld, Owner: %s\n", dp->d_name, info.st_size, ps->pw_name) ;

            strcpy(path, base_path);
            strcat(path, "/");
            strcat(path, dp->d_name);
            list_dir(path, root + 2);
        usleep(100000); // to make it look coooool!
        }
    }

    closedir(dir);
}
```