CS 332/532 – 1G- Systems Programming

Lab 5

Objectives

The objective of this lab is to introduce you to UNIX file system and properties of files.

Description

We'll start with the *stat* and *lstat* functions. Both functions return a structure called *stat*, and members of stat structure provide information about the file or directory which was provided as the argument to these functions. Please refer to Section 4.2 in the book or lecture 12 and learn more about *stat* structure, also run *man stat* command in terminal.

Example 1:

Now let's write program to print the type of file for any given command-line argument (you can refer to Figure 4.3 in the book). We use the macros from Figure 4.1 from the textbook to determine the file type. You can download the files: `lstat.c` and `printstat.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

void printstat(struct stat statbuf);

int main(int argc, char **argv) {
    int i;
    struct stat buf;
    char *ptr;

    for (i = 1; i < argc; i++) {
        printf("%s: ", argv[i]);
        if (lstat(argv[i], &buf) < 0) {
            printf("lstat error");
            continue;
```

```
        }

        if (S_ISREG(buf.st_mode))
            ptr = "regular";
        else if (S_ISDIR(buf.st_mode))
            ptr = "directory";
        else if (S_ISCHR(buf.st_mode))
            ptr = "character special";
        else if (S_ISBLK(buf.st_mode))
            ptr = "block special";
        else if (S_ISFIFO(buf.st_mode))
            ptr = "fifo";
        else if (S_ISLNK(buf.st_mode))
            ptr = "symbolic link";
        else if (S_ISSOCK(buf.st_mode))
            ptr = "socket";
        else
            ptr = "** unknown mode **";
        printf("%s\n", ptr);
    }
    printstat(buf);

    exit(0);
}
```

Above program we have use lstat function, which is similar to stat function, except that when there is a symbolic link the lstat function provides details about the link itself not the file the link points to.

Example 2:

Till now we talked about how filesystem store files and directories information and access details. Now it's time to learn how to open and read the directories and traverse the file system. To achieve this task, you need to learn about three functions:

1. *opendir* - this function will allow us to open a directory with the given path
2. *readdir* - this function will read what's inside the directory
3. *closedir* - this will close the open directory

Now let's write a simple program to open, read, and close a directory (read section 4.22 in book). You can download the program here: `readdir.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
```

```
int main (int argc, char **argv) {
  struct dirent *dirent;
  DIR *parentDir;

  if (argc < 2) {
    printf ("Usage: %s <dirname>\n", argv[0]);
    exit(-1);
  }

  parentDir = opendir (argv[1]);
  if (parentDir == NULL) {
    printf ("Error opening directory '%s'\n", argv[1]);
    exit (-1);
  }

  int count = 1;
  while((dirent = readdir(parentDir)) != NULL){
    printf ("[%d] %s\n", count, (*dirent).d_name);
    count++;
  }

  closedir (parentDir);
  return 0;
}
```

Here is a slightly modified version of the above program that prints the file type next to the filename: `readdir_v2.c`

We can use the read/write system calls from Lab-04 to write the stat structure and read the stat structure. Note that data is written as a binary file. The following files illustrate how to write the stat structure and read the stat structure (you have compile with the `printstat.c` file as shown below): `readstat.c` and `writestat.c`

```
$ gcc -Wall writestat.c printstat.c
$ ./a.out lstat.c stat.out
File type: regular file
I-node number: 8609355647
Mode: 100755 (octal)
Link count: 1
Ownership: UID=503 GID=20
Preferred I/O block size: 4096 bytes
File size: 1042 bytes
Blocks allocated: 8
Last status change: Wed Feb 12 18:04:51 2020
Last file access: Wed Feb 12 18:04:54 2020
Last file modification: Wed Feb 12 18:04:51 2020
```

```
$ gcc -Wall readstat.c printstat.c
$ ./a.out stat.out
File type: regular file
I-node number: 8609355647
Mode: 100755 (octal)
Link count: 1
Ownership: UID=503 GID=20
Preferred I/O block size: 4096 bytes
File size: 1042 bytes
Blocks allocated: 8
Last status change: Wed Feb 12 18:04:51 2020
Last file access: Wed Feb 12 18:04:54 2020
Last file modification: Wed Feb 12 18:04:51 2020
$
```

You can find a more elaborate example in Figure 4.22 in the textbook. This program takes as input a directory name, traverses a file hierarchy, counts the different types of files in the given file hierarchy, and the prints the summary (as shown in Figure 4.4). This program uses function pointers i.e., you can pass a function as an argument to a function similar to passing variables of different type. This enables us to perform different operations on a file as we traverse the file hierarchy. Here is a simple example that illustrates the use of function pointers: `funcptr.c`

```
/* Sample program to illustrate how to use function pointers */
#include <stdio.h>

typedef int MYFUNC(int a, int b);

int add(int a, int b) {
    printf("This is the add function\n");
    return a + b;
}

int sub(int a, int b) {
    printf("This is the subtraction function\n");
    return a - b;
}

int opfunc(int a, int b, MYFUNC *f) {
    return f(a, b);
}

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

    int a = 10, b = 5;
    printf("Passing add function....\n");
    printf("Result = %d\n", opfunc(a, b, add));
```

```
    printf("Passing sub function....\n");
    printf("Result = %d\n", opfunc(a, b, sub));

    return 0;
}
```

In this example we define a function *opfunc* that takes as input a pointer to a function that takes two integer arguments and returns an integer value. We use *typedef* to define the function signature so that we can use this as a type in the function definition. Then we can have different functions with the given type signature and these functions can perform different operations. In this example, we define two functions to perform addition and subtraction on the two arguments passed to the function. Now we can invoke the *opfunc* by providing two integer values and the corresponding function to perform the required operation.

Example 4.22 uses this mechanism through which we can define different functions to perform different operations on a given file as we traverse the file hierarchy. In this example, we just count the different files types, however, we could perform other operations such as check the file size or file permission or any other user-defined operation.

Homework

Extend the readdir program to traverse the file hierarchy recursively and list all the sub directories and the files in these sub directories. Note that you have to modify the given readdir.c (or readdir_v2.c) program to use a function to perform the file traversal and then invoke it recursively.

Submission

You are required to submit the lab solution to Canvas

1. To upload solution to Canvas: Upload the C source code, PDF version of C source code, and README.txt file to Canvas. Do not include any executable or object files.

Use the following command to create PDFs of your source code.(replace the <Source_code_File_name> and <Output_Source_code_File_name> with your C source code file name):

```
$enscript <Source_code_File_Name>.c -o - | ps2pdf - <Output_Source_code_File_Name>.pdf
```