Project 3

Report

Course Name: BLG413E – System Programming

Date: 13.12.2017

Group Information		
Student ID	**Name**	**CRN**
150140119	Muhammed Kadir Yücel	13331
150140123	*Mahmut Lutfullah Özbilen*	*13332*

* This report was prepared by Muhammed Kadir Yücel *(150140119)*

Table of Contents

Introduction

This report was prepared for Project 3 of System Programming course by Muhammed Kadir Yücel *(150140119)*. In this project we aimed to create a file system on user level that creates hierarchy from a CSV(Comma Separated Values) file input. After reading CSV file a file system should be mounted to the system and users would be able to see files/folders inside it. Also users can rename or delete files inside file system and changes will be represented on CSV file. In addition to that users can also be able to update CSV file and changes will be seen on the file system immediately.

We have used FUSE library to implement a file system at user level and CSV library to work on CSV file. Source file and example input CSV file can be found inside *.zip file provided through Ninova system.

Compiling

There are some libraries required to compile and use this code;

1. FUSE library (libfuse)

```
sudo apt-get install libfuse-dev
```

2. CSV library (libcsv)

```
sudo apt-get install libcsv-dev
```

After installing above libraries to the system you can continue compiling the *main.c* which is the source code of the program,

1. Extract provided *.zip file inside a folder
2. Open that folder inside terminal
3. `gcc main.c -o main -D_FILE_OFFSET_BITS=64 -lfuse -lcsv`

Running

You should have *postal-codes.csv* file near with the compiled executable and an empty folder in the system that your user can have permissions on it.

1. Make sure that *postal-codes.csv* file is in the same directory with the executable file.
2. `mkdir test`
3. `./main test` for standard running
4. `./main -d test` for debug mode
5. Now when you open the *test* folder you will see that the file system is attached and working

Project Work

CSV File Operations

We have used *libcsv* to open and read the CSV file. We have defined a structure to define CSV file's row in our program.

```
// START: CSV things
typedef struct csv_row {
        char* code;
        char* neighborhood;
        char* city;
        char* district;
        char* latitude;
        char* longitude;
        struct csv_row* next;
} crow;
```

Image 1: Data Structure for CSV Row

We have created a linked list by using above structure as node in the linked list to hold each row and represent it on the file system. Since we have used an existing CSV library, that library needed some function definitions in the program to work properly. After opening CSV file and sending the file pointer into the *csv_parse* function of the library, library calls some functions for each row and each column in the file. We've defined column separator as *\t* character and row separator as *\n* character by using library's *csv_set_delim* function.

```
csv_set_delim(&p, '\t');
fp = fopen("postal-codes.csv", "rb");
if(!fp){
        printf("Failed to open file %s\n", strerror(errno));
        exit(EXIT_FAILURE);
}

while((bytes_read = fread(buf, 1, 1024, fp)) > 0){
        size_t ret = csv_parse(&p, buf, bytes_read, cb1, cb2, &my_list);
        if(ret != bytes_read){
                printf("Error while parsing file :%s\n", csv_strerror(csv_error(&p)));
        }
}
```

Image 2: Openning and reading CSV file into linked list

For each column in the file library calls *cb1* function and for each row library calls *cb2* function in the *main* source code file. In *cb1* function we counted columns and put it into the variable's required variables which is the type of *crow*. In *cb2* function we added new empty node to the linked list to represent new row.

FUSE Operations

We've used following operations in FUSE to create our file system with the properties wanted in project's requirements document;

- getattr
  This function is an important and required part of the FUSE. It returns attributes of the files that will be represented on file system. We set files as file or folder by setting required attributes.
- readdir
  When user opens a directory inside file system, this function will be called to fill inside the folder with required files or folders. It returns directory entries.
- open
  When user opens a file this function will be called to get file contents.
- read
  After open function called, this function will be called to write file's content ot the buffer memory that will be read by the opened program that reads the file.
- rename
  Renaming a directory or file calls this function, after renaming we got the old and new value (file name) and updated matching value in linked list and updated CSV file.
- unlink
  When user deletes a file on file system this function will be called and we have implemented that when we get the deleted file's information we searched through linked list and deleted that row and updated CSV file according to the new linked list.

```
static struct fuse_operations fuse_oper = {
        .getattr = fuse_getattr,
        .readdir = fuse_readdir,
        .open = fuse_open,
        .read = fuse_read,
        .rename = fuse_rename,
        .unlink = fuse_unlink,
};
```

Image 3: Structure that holds functions to be called for FUSE operations

fuse_getattr

This function takes two arguments, *path* and a buffer memory to hold file flags. Since our project has fixed structure we set values according to '/' character inside *path* value. When user enters file system's mounted folder this function will be called with *path* with value '/', so we set that contents

of this directory is a folder. But if *path* value contains "CODES" and 3 of the '/' characters, for example *CODES/34/34398.txt*, we set the flags in a way that that path will represent a file.

```
stbuf->st_mode = S_IFDIR | 0755;
stbuf->st_nlink = 2;
```

Image 4: Directory representation

But we have to specify file size in this function while setting flags to open and read file's contents properly. Since files are simple txt files that will contain character information, we have searched required information through linked list and calculated file size inside here.

```
stbuf->st_mode = S_IFREG | 0664;
stbuf->st_nlink = 1;

char* token_path = (char*)malloc(strlen(path)+1);
strcpy(token_path, path);
char* token = strtok(token_path, "/");

char* file_name;
char* plate;
int i = 0;
while(token != NULL){
        i++;
        if(i == 2)
                plate = token;
        if(i == 3)
                file_name = token;

        token = strtok(NULL, "/");
}

char* code = strtok(file_name, ".");

size_t file_length = 0;
crow *temp = my_list.head;
while(temp != NULL){
        if(temp->code == NULL){
                temp = temp->next;
                continue;
        }

        if(strcmp(temp->code, code) == 0 ){


                file_length = strlen(fcode) + strlen(fneighborhood) +
                        strlen(fcity) + strlen(fdistrict) +
                        strlen(flatitude) + strlen(flongitude) +
                        strlen(temp->code) + strlen(temp->neighborhood) +
                        strlen(temp->city) + strlen(temp->district) +
                        strlen(temp->latitude) + strlen(temp->longitude) + 1;
                break;
        }
        temp = temp->next;
}
printf("LENGTH IS %d\n", file_length);
stbuf->st_size = file_length;
```

Image 5: File setting and defining file size

fuse_readdir

This function called when a user wants to read a directory for example in a GUI file manager when user double clicks on a folder and wants open that folder, this function will be called, in terminal when user *cd* into a folder and calls *ls* program this function will be called. Inside this function we called *filler* function to fill directories, again according to the number of '/' characters inside *path* value. But since in the CSV file we have many rows with the repeating city names, for example many 'Adiyaman' directories may be represented on the file system, so we hold another linked list

and added filled values to that linked list and checked that linked list if this value added before or not.

```
filler(buf, ".", NULL, 0);
filler(buf, "..", NULL , 0);
filler(buf, names_path+1, NULL, 0);
filler(buf, codes_path+1, NULL, 0);
```

Image 6: Filling values inside directories

fuse_open

This function is called while opening a file or directory in file system. In this function we've only checked that if the file or directory has the privilege of opening flag set.

fuse_read

This function called when a file open operation called from the file system. This function is called when user wants to open a .txt file inside file system, in this state we put the required values inside buffer memory to show the data inside text file reader. In this time data that will be put on the buffer has to have same size with the size defined for file in the *fuse_getattr* function.

fuse_unlink

This function called when user wants to delete a file in file system. This function gets the path of the deleted file, in this function we extracted file name from the path and searched it through the linked list and deleted that item from linked list. After updating linked list we write updated linked list to the CSV file.

fuse_rename

This function called when a user wants to rename directory or file. We only checked for file renaming and directory renaming is not allowed. This function gets two arguments which are *from* and *to*. *From* argument contains path with old file name and *to* argument contains new path with the new file name so after we found old file's representing node in linked list, we updated its name and write linked list into the CSV file.

Conclusion

In this project we've learned;

- How to implement user level file systems
- FUSE library
- CSV library
- Reading a CSV file
- File system operations

- File flags and privileges

With this project we gained a skill that we can implement file systems. User level file systems are really popular in today's word increasing cloud storage solutions. For example cloud storage solutions like Google Drive, OneDrive generally provides a program that you can see your cloud storage in your local machine's file manager.