

OPERATING SYSTEMS(IT253)

MINI PROJECT REPORT



Faculty: Dr. Neelima Bayappu.

Year: 2nd Year B.Tech.

Department of Information Technology,
National Institute of Technology Karnataka, Surathkal
Mangaluru, India.

Submitted by :

Amith Bhat Nekkare	(181IT105)
Asis Rout	(181IT108)
Ashok Bhobhiya	(181IT154)
Jeeukrishnan Kayshyap	(181IT220)

June 2020

TABLE OF CONTENTS

1. Table of Contents	2
2. Abstract	3
3. Introduction	3
4. Objectives	4
5. Methodology	4
6. Functionalities	5
7. Implementation Details	6
8. Output & Results	12
9. Challenge Faced	13
10. Conclusion	14

Abstract:

In this project, we will implement a file system that supports access to a File Transfer Protocol (FTP) using FUSE, and add support for commands such as ls, cat, etc.

Introduction:

A Filesystem In Userspace (FUSE) is a computer program that allows the user to implement a file and directory structure of the user's own creation, which works just as if it were a local file structure. The FUSE system has been used by programmers for a variety of purposes and forms the basis of many of the ways of accessing non-local-files in the modern Fedora and Ubuntu desktops.

FUSE systems are basically “enablers”, ie, they allow the development of other applications. For example, virtual filesystems are mainly written using FUSE filesystems, as there is no need to write kernel code. FUSE works on the basic principle of “anything as a file”, allowing operations on and access to a diverse set of files and directories. FUSE calls are also quite similar to standard calls. Other advantages of FUSE are its efficiency in storage and the fact that any user can write their files, can mount it, and can get it shared with other people also.

In this project, we are implementing the file sharing protocol, FTP in a FUSE filesystem. This would allow for easier transfer of files from one user to another. In Linux systems, FTPFS (FTP using File Systems) was initially implemented as a module that allows the user to mount an FTP server onto the local filesystem.

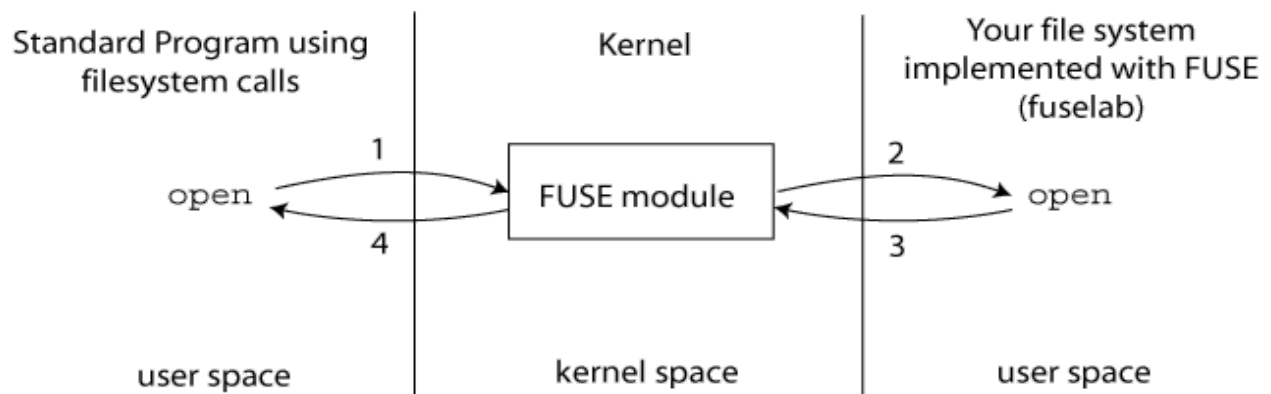


Fig 1: A Pictorial Representation of the working of FUSE

1. A program, such as ls, mkdir, or perflab makes a call to a file system routine. For example, open("/test/fuse/file1.txt"). This call gets sent to the kernel.
2. If this file is in a FUSE volume, the kernel passes it on to the FUSE kernel module, which then passes it on to the implementation of that file system (this is the portion we will be writing in this lab).
3. The implementation of open then refers to the actual data structures that represent the file system and returns a file handle. It is the open's job to take a concrete view of data (bits stored on a hard drive) and present an abstract view (a hierarchically organized file system).
4. The kernel returns the result of the open function to the program that originally made the call.

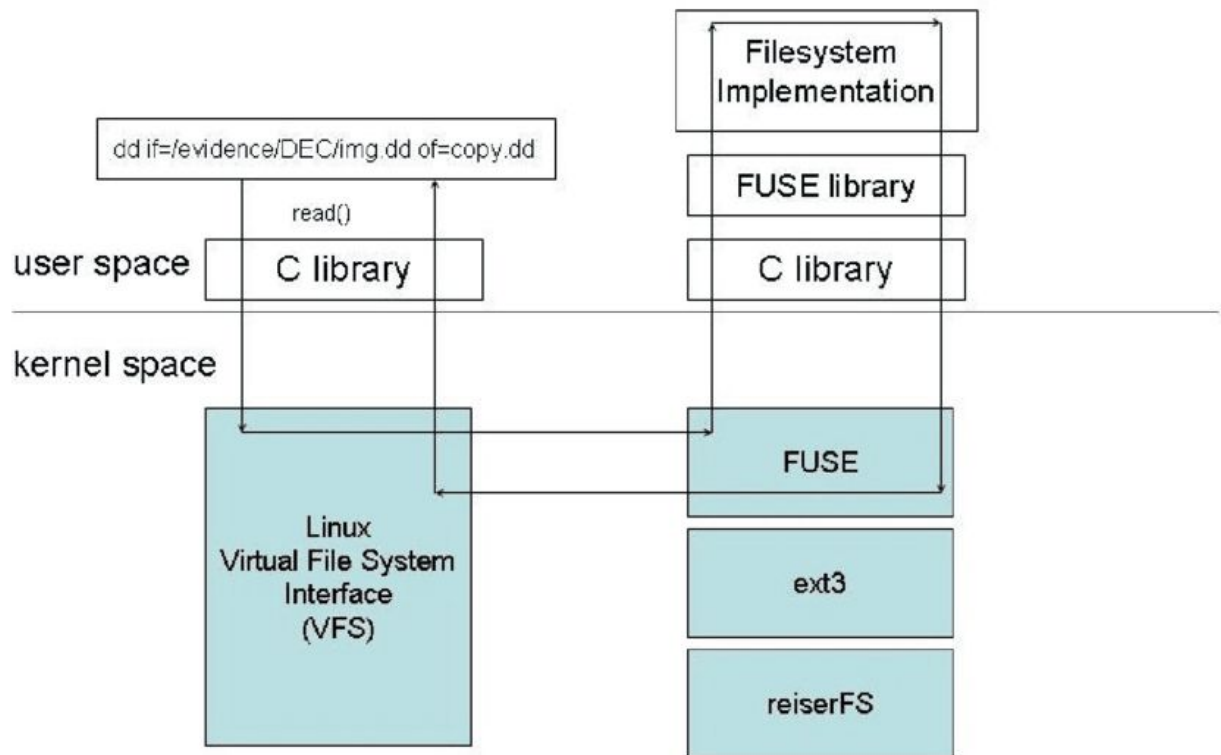
Objectives:

1. To implement a FUSE filesystem in a Linux machine.
2. To implement file transfer protocols such as FTP in a FUSE filesystem.
3. To implement some Linux commands such as ls, cat,etc., in the FUSE filesystem.

Methodology:

A. Implementation of File System Using FUSE

1. Install FUSE.
2. Make the structure of FUSE.
3. Write the function “getattr” - it returns information about each file.
4. Write the function “readdir”- it lists the files/directories which are available.
5. Write the “read” function to read a specific file.



B. Implementation of FTP:

1.Implementation of FTP in FUSE.

C. ADD commands :

1. Add some new commands to the system such as ls, cat, etc.

Functionalities of FUSE filesystem:

FUSE filesystems provide all facilities without user root privilege. Any user can write their files, can mount it, and can get it shared with other people also. Useful to develop “virtual” filesystems. Using FUSE filesystems, there is no need to write kernel code. It allows you to write a secure file system that can be mounted by him/her. FUSE file systems allow you to creatively assume “anything as a file”, hence allowing file operation on remote directories. We can also extract any type of database, such as SQL databases. The calls are very similar to standard calls, like `open(path)`, `create(path, mode)`, etc. It is also a storage efficient filesystem. It allows us to develop filesystems in any language like C/C++, Java, Python.

Implementation Details ::

Basically there are four files on the project :

- 1.fuseftp.hpp (header file for fuse)
- 2.fuseftp.cpp
- 3.ftpcient.hpp (header file for ftp)
- 4.ftpcient.cpp

a)fuseftp.cpp

Code snippet ::

```
static int fuse_getattr (const char *, struct stat *);

static int fuse_readdir(const char *path, void *buf,
                        fuse_fill_dir_t filler,
                        off_t offset, struct fuse_file_info *fi
                        );

static int fuse_read(const char *path, char *buf, size_t size, off_t offset,
                    struct fuse_file_info *fi);
```

So we implement a basic FUSE filesystem. The code contains basically three functions:

1.“getattr” :

- essential to write a functional filesystem.

```
static int fuse_getattr( const char *path, struct stat *stbuf)
```

First parameter is the path of the file which the system asked for its attributes. The other parameter is “stat” which must be filled with the attributes of the file.

```

if ( strcmp( path, "/" ) == 0 )
{
    st->st_mode = S_IFDIR | 0755;
    st->st_nlink = 2;
}

```

The field “st_mode” specifies if the file is a regular file, directory or other.

The field “st_nlink” specifies the number of hardlinks.

the field “st_size” specifies the size of that file in bytes.

In the above part of the code we fill “st_mode” with the macro S_IFDIR which indicates that the file in question is a directory. then we set the permission bits as the following: only the owner of the file could read, write and execute the directory, the group’s users and other users could only read and execute the directory

After filling “st_mode” for the root directory we specify the number of hardlinks,

```

else
{
    st->st_mode = S_IFREG | 0644;
    st->st_nlink = 1;
    st->st_size = 1024;
}

return 0;

```

In “st_mode” for those files we use the macro `S_IFREG` to indicate that they are just regular files, the owner could read and write the file, the group’s users and other users could only read the file.

2.“readdir” :

In readdir we could list the files/directories which are available inside a specific directory

```
static int fuse_readdir( const char *path, void *buf, fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fi )
```

Parameters :

path : path of the directory

buffer : we are going to fill the names of the files/directories which are available inside the directory

Filler : function sent by FUSE and we could use it to fill the “buffer” with available files in “path”. The function returns 0 on success.

3.“read” :

Through this function the system could read the content of a specific file.

```
static int fuse_read( const char *path, char *buf, size_t size, off_t offset, struct fuse_file_info *fi )
```

Parameters :

Path : path of the file which the system wants to read.

Buffer : we are going to store the *chunk* which the system interested in,

Size : size” represents the size of this chunk

Offset : “offset” is the place in the file’s content where we are going to start reading from.

“fuse_read” must return the number of the bytes that have been read successfully.

b)ftpclient.cpp

Code snippet :

```
static int url_fclose(URL_FILE *file);
static int url_feof(URL_FILE *file);
static size_t url_fread(void *ptr, size_t size, size_t nmemb, URL_FILE *file);
static char *url_fgets(char *ptr, size_t size, URL_FILE *file);
static void url_rewind(URL_FILE *file);
static size_t write_callback(char *buffer,
                             size_t size,
                             size_t nitems,
                             void *userp);

static int fill_buffer(URL_FILE *file, size_t want);
static int use_buffer(URL_FILE *file, size_t want);
static size_t GetFilesList_response(void *ptr, size_t size, size_t nmemb, void *data);
```

Here we used libcurl for our ftpclient.

int fill_buffer :

use to attempt to fill the read buffer up to requested number of bytes

int use_buffer :

use to remove want bytes from the front of a files buffer

size_t GetFilesList_response :

Use to get the list of files

int url_fclose :

Close the file in the url.

int url_feof :

Test the end of the file in the url.

size_t url_fread :

Read block of data from the file in the url.

char *url_fgets :

Read characters from the file in the url.

size_t write_callback:

curl calls this routine to get more data.

url_rewind :

the rewind function sets the file position to the beginning of the file for the stream pointed to by stream.

So ,basically in this part, the file is taken from the url if it is a file then you can read the file using the buffer , else if it is a directory you can get the list of files.

Now the read function in the directory(i.e. cat command) fails and listing of files (i.e. ls command) in the files fails.

****Connection with FTP Server and Fuse:**

Code snippet :

```
handle = url_fopen(path, "r");
if(!handle) {
    cout<<"Couldn't url_fopen()\n";
    return 2;
}
return 1;
```

So , the connection is done in the fuse.cpp file itself.Here we are passing the url as a parameter to url_fopen(). If the return value is not NULL, then the connection is successful else not.

```
char buffer[256];
string cont;
while(!Fc.url_feof(handle)) {

    Fc.url_fgets(buffer, sizeof(buffer),
handle);
```

```

    cont += buffer;
}
options.contents = cont.c_str() ;

```

This above code is used to read the contents of url using `url_fgets()` and store it in member contents of class options.

```

void CloseFTPConnection()
{
    Fc.url_fclose(handle);
}

```

At last, the connection to the server is closed by using `url_fclose()`.

```

bool isDirectory(string url) //Check if URL is a Directory or not (ftp://192.168.1.100:2121//
{
    if(url[url.length()-1]=='/')
        return true;
    return false;
}

bool isFile(string url) //Check if URL is a file or not (ftp://192.168.1.100:2121/testfile.txt/*)
{
    if(url[url.length()-1]=='*')
        return true;
    return false;
}

```

Depending on whether the url is directory or file, different methods would be called by fuse. The customs applied to differentiate between the directory and file is shown above.

- If the url is a directory
 - ls : It would list all folders and files in the directory
 - cat : This would give an error message since the url is directory and cat won't work.
- If the url is a file
 - ls : This would give an error message since the url is a file and ls won't work.
 - cat : This would print the contents of the file.

Instruction to Run :

Compile the file: `g++ -Wall *.cpp -lcurl `pkg-config fuse --cflags --libs` -o fuseftp`

Mounting the filesystem: `./fuseftp ftp://192.168.1.100:2121// test (Mount test directory)`

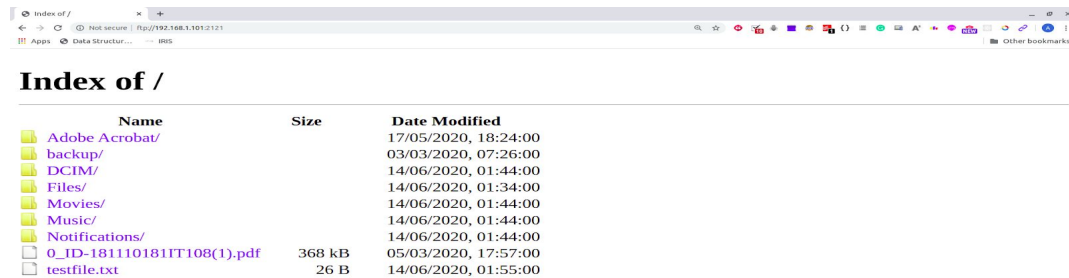
List Directory: `ls test`

Read the content from file: `cat test/file`

Output and Results :

Screenshots:

- Files present in FTP Server:

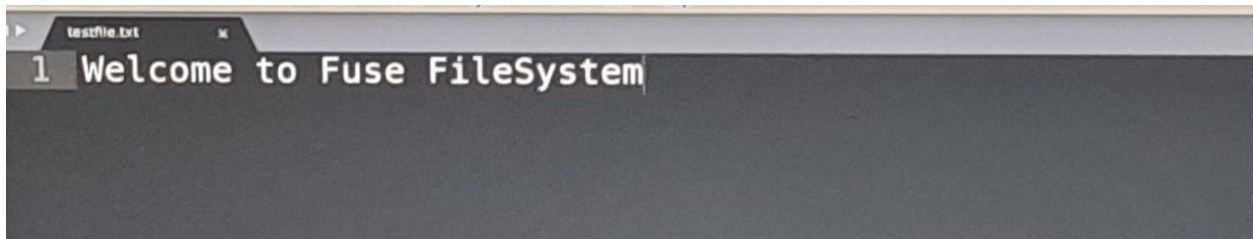


Name	Size	Date Modified
Adobe Acrobat/		17/05/2020, 18:24:00
backup/		03/03/2020, 07:26:00
DCIM/		14/06/2020, 01:44:00
Files/		14/06/2020, 01:34:00
Movies/		14/06/2020, 01:44:00
Music/		14/06/2020, 01:44:00
Notifications/		14/06/2020, 01:44:00
0_ID-181110181IT108(1).pdf	368 kB	05/03/2020, 17:57:00
testfile.txt	26 B	14/06/2020, 01:55:00

- Listing the Directories present in Ftp server using ls command in fuse filesystem

```
asis@ubuntu: ~/Desktop/OS/FinalProject/Project
asis@ubuntu:~/Desktop/OS/FinalProject/Project$ ./fuseftp ftp://192.168.1.101:2121/ test
asis@ubuntu:~/Desktop/OS/FinalProject/Project$ ls test
'dr-x----- 3 anonymous anonymous 0 May 17 12:54 Adobe Acrobat'$\n'$dr-x----- 3 anonymous anonymous 0 Mar 3 01:56 back
up'$\n'$dr-x----- 3 anonymous anonymous 0 Jun 13 20:14 DCIM'$\n'$dr-x----- 3 anonymous anonymous 0 Jun 13 20:04 Files
'$\n'$dr-x----- 3 anonymous anonymous 0 Jun 13 20:14 Movies'$\n'$dr-x----- 3 anonymous anonymous 0 Jun 13 20:14 Music
'$\n'$dr-x----- 3 anonymous anonymous 0 Jun 13 20:14 Notifications'$\n'$-r----- 1 anonymous anonymous 376376 Mar 5 12:2
7 0_ID-18110181IT108(1).pdf'$\n'$-r----- 1 anonymous anonymous 26 Jun 13 20:25 testfile.txt'$\n'$
asis@ubuntu:~/Desktop/OS/FinalProject/Project$ cat test/file
Error:The URL is a directory not a file
asis@ubuntu:~/Desktop/OS/FinalProject/Project$
```

- Contents in testfile.txt in FTP Server



- Using cat command to read file

```
asis@ubuntu: ~/Desktop/OS/FinalProject/Project
asis@ubuntu:~/Desktop/OS/FinalProject/Project$ ./fuseftp ftp://192.168.1.103:2121/testfile.txt/* test
asis@ubuntu:~/Desktop/OS/FinalProject/Project$ cat test/file
Welcome to Fuse FileSystem
asis@ubuntu:~/Desktop/OS/FinalProject/Project$ ls test
'Error:The URL is a file not a directory'
asis@ubuntu:~/Desktop/OS/FinalProject/Project$ fusermount -u test
asis@ubuntu:~/Desktop/OS/FinalProject/Project$
```

Challenges Faced:

1. Coding for the FTP client was a challenge, considering there exists very little resources to write it in C++, the language in which we have coded. Especially, there are no readily-available libraries for this task in C++.
2. Due to specificities of teammates' computers and operating systems, FUSE installations were also an issue.

Conclusion:

In our project, we have successfully :-

1. Implemented the FTP protocol in a FUSE filesystem, in C++. The C++ implementation of this does not exist on the internet, hence it is quite possible for this project to be unique in that regard.
2. Implemented some simple commands of Linux terminal like ls, cat, mount, etc. In our FUSE filesystem.