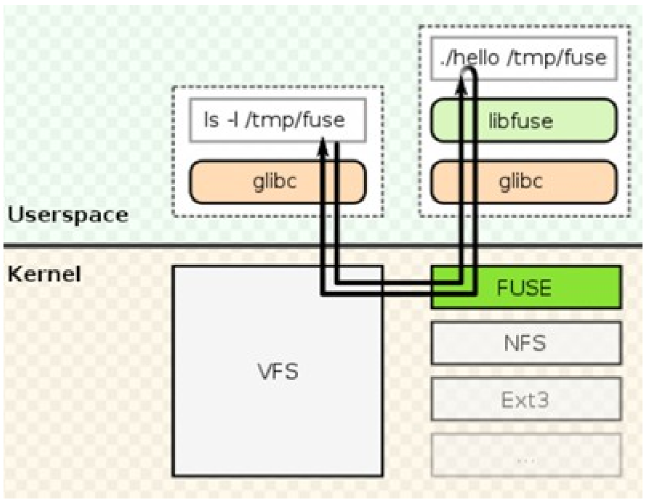
FILE SYSTEM USING FUSE

CSE

Rakesh Reddy Bijjam
Prajwal S

ABOUT FUSE

Filesystem in Userspace (FUSE) is a software interface for Unix-like computer operating systems that lets non-privileged users create their own file systems without editing kernel code. This is achieved by running file system code in user space while the FUSE module provides only a "bridge" to the actual kernel interfaces.



courtsey wikipedia

To implement a new file system, a handler program linked to the supplied “libfuse” library needs to be written. The main purpose of this program is to specify how the file system is to respond to read/write/stat requests. The program is also used to mount the new file system. At the time the file system is mounted, the handler is registered with the kernel. If a user now issues read/write/stat requests for this newly mounted file system, the kernel forwards these IO-requests to the handler and then sends the handler's response back to the user.

FUSE is particularly useful for writing virtual file systems. Unlike traditional file systems that essentially work with data on mass storage, virtual filesystems don't actually store data themselves. They act as a view or translation of an existing file system or storage device.

The rationale of implementing a Filesystem in userspace apart from giving you more control over the file operations , it also allows you to achieve more portability without implementing a kernel module and can easily be ported even via containerization mechanisms. Also a bug wont bring the kernel down as this is completely a userspace implementation.

ABOUT THE PROJECT

To implement filesystem using fuse. Allowing creation of files and directories and perform basic operation.

Goals

1) To understand the concepts of file systems in Linux.
2) To gain practical knowledge of Linux internals by creating a basic filesystem using tree structure to simulate hierarchical representation

Features

- Facilitate users to create files and directories.
- Enable users to list all the contents of the directories.
- Allow users to write into a file.
- Allow users to read the file.
- Users can remove the files or directories which are not of need.
- Persistence of filesystem also allows users to work on files over time.

Project phases

Phase 1: System call Implementation

Implementation of system calls for file operations that can be issued by Linux using FUSE.System calls implemented are – ls, mkdir, rmdir,echo, cat, rmdir, rm.

Phase 2: File System Abstraction

This involves development of internal data structures and procedures necessary to implement files. Abstraction helps in showing information that is required by the user and not showing any unnecessary details. Abstraction is done with the help of Inodes and superblocks.

Phase 3: Secondary Storage(Persistence)

This was achieved by keeping track of the mounted file system using structures. These structures and their corresponding changes were written to a binary file. When the file system was mounted, the binary file was read, and the structures implemented before handing it out to the user. This resulted in a persistent file system where-in changes made would not be lost upon unmounting

Hardware Requirement

FUSE is available for

- Linux
- FreeBSD
- OpenBSD
- NetBSD
- OpenSolaris
- Minix 3
- Android
- macOS.

Software Requirement

FUSE is free software originally released under the terms of the GNU General Public License and the GNU Lesser General Public License.

FUSE SETUP

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

File System Design

Structure

- Size of datablock – 512 bytes
- Maximum number of data blocks – 15

- Data type struct is used for file, file-info, directory, block, superblock and root information storage

Description

- Tree based approach to keep track of all the files. First we initailize the superblocks (number of blocks in total and their status).
- Mkdir – Initialization of directory contents using struct directory. It is linked to the parent node . Every new directory in the same level is appended to a linked list .
- Remove Directory – It involves finding the directory using the path provided and deleting it from the directory linked list .
- Create File - Traversal is done from the root node to the path. If the file doesn’t exist it is created. struct file and fileinfo is used to store all the required information about the file.
- Write File - Traversal is done from the root node to the path. Struct block is used to keep track of data. Struct file and fileinfo is used to store all the required information about the file.

Implementation

Core Functions

- .readdir(self, path, offset) as file_readdir.
- .getattr(self, path) as file_getattr
- .mkdir(self, path, mode, dev) as file_mkdir
- .open(self, path, flags) as file_open
- .read(self, path, size, offset) as file_read
- .write(self, path, buf, offset) as file_write
- .create(self,path,flags) as file_create
- .destroy(self) as file_destroy
- .rmdir(self, path) as file_rmdir

- .unlink(self, path) as file_unlink

Persistence

Theory

Persistent storage is any data storage device that retains data after power to that device is shut off. Hard disk drives and solid-state drives are common types of persistent storage. This can be in the form of file, block or object storage.

Binary files are usually thought of as being a sequence of bytes, which means the binary digits (bits) are grouped in eights. Binary files typically contain bytes that are intended to be interpreted as something other than text characters. Compiled computer programs are typical examples

Approach

I. Storing after unmount

- Open the binary file (restore) to append information .
- Store the information about the root node .
- Store all the directories' information along with information of it's contents .
- Store all the file information from the root node.
- Store the contents in the super block.

II. Restoring after mounting

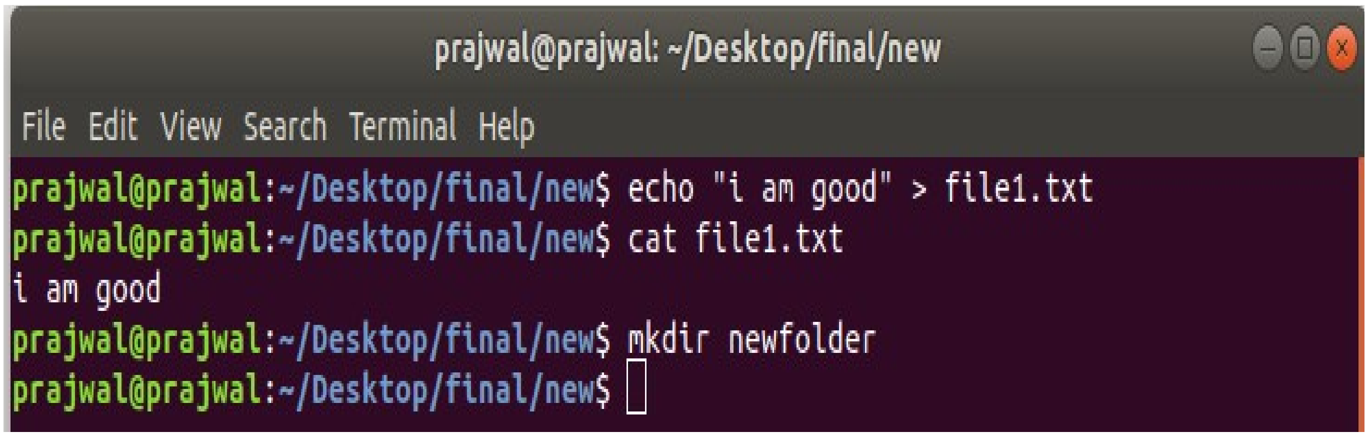
- Open the binary file (restore) to read information.
- Read the root node information to restore the root.
- Read the super block node information .
- Read the files in the root node .

Execution

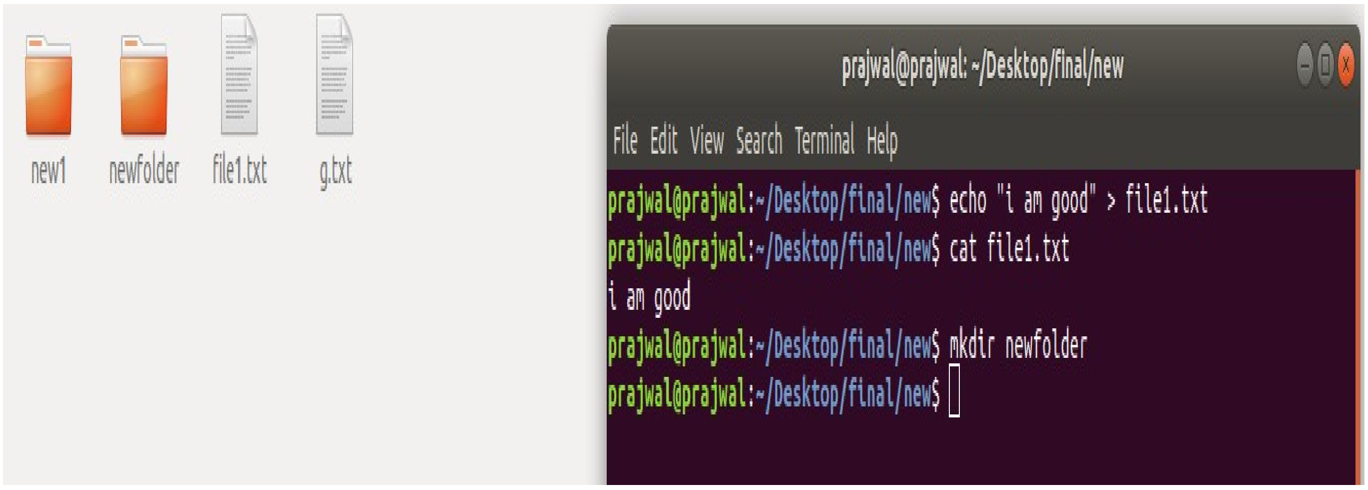
```
gcc -Wall -w fs.c `pkg-config fuse --cflags --libs` -o fs
```

```
./fs newmount –f
```

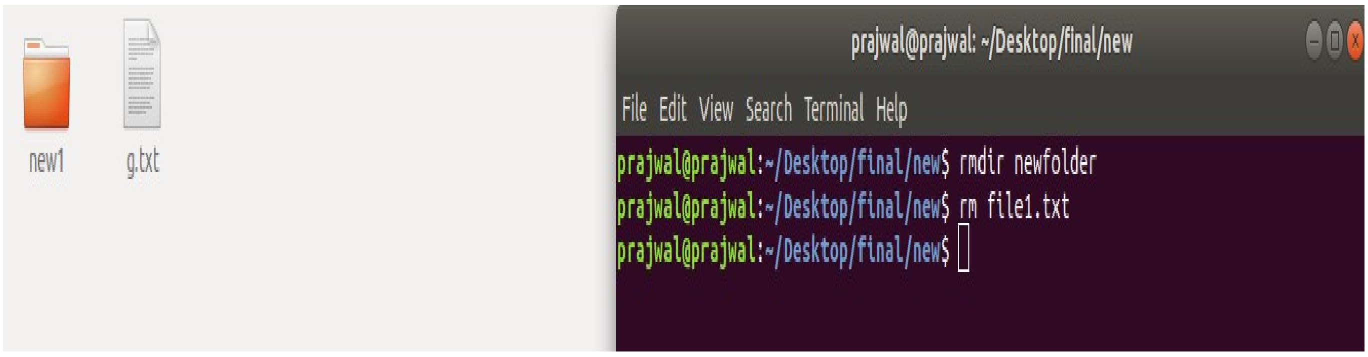
Screenshots



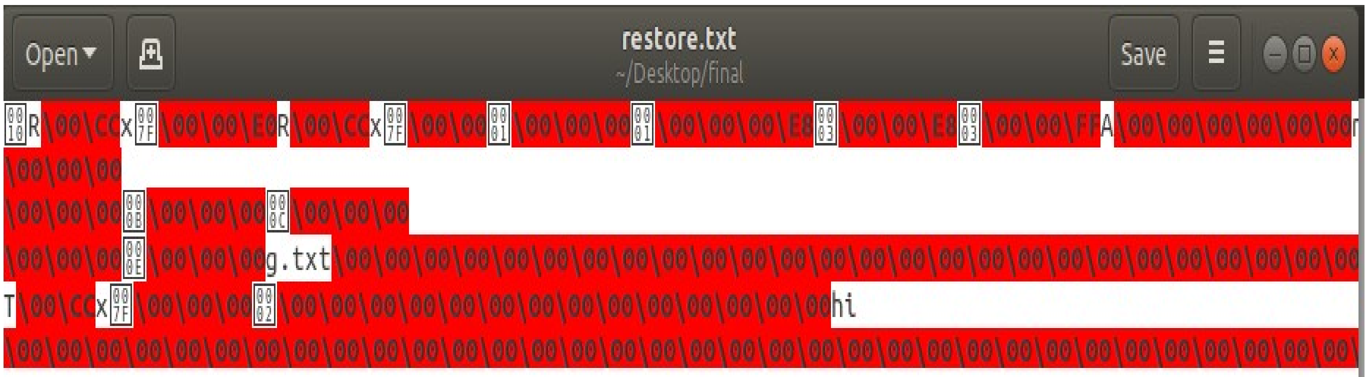
```
prajwal@prajwal:~/Desktop/final/new$ echo "i am good" > file1.txt
prajwal@prajwal:~/Desktop/final/new$ cat file1.txt
i am good
prajwal@prajwal:~/Desktop/final/new$ mkdir newfolder
prajwal@prajwal:~/Desktop/final/new$ 
```



```
prajwal@prajwal:~/Desktop/final/new$ echo "i am good" > file1.txt
prajwal@prajwal:~/Desktop/final/new$ cat file1.txt
i am good
prajwal@prajwal:~/Desktop/final/new$ mkdir newfolder
prajwal@prajwal:~/Desktop/final/new$ 
```



```
prajwal@prajwal:~/Desktop/final/new$ rmdir newfolder
prajwal@prajwal:~/Desktop/final/new$ rm file1.txt
prajwal@prajwal:~/Desktop/final/new$ 
```



```
0010R\00\CCx007F\00\00\E0R\00\CCx007F\00\000001\00\00\000001\00\00\00\E80003\00\00\E80003\00\00\FFA\00\00\00\00\00\00
\00\00\00
\00\00\00000B\00\00\00000C\00\00\00
\00\00\00000E\00\00\00g.txt\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
T\00\CCx007F\00\00\000002\00\00\00\00\00\00\00\00\00\00\00\00\00\00hi
\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
```