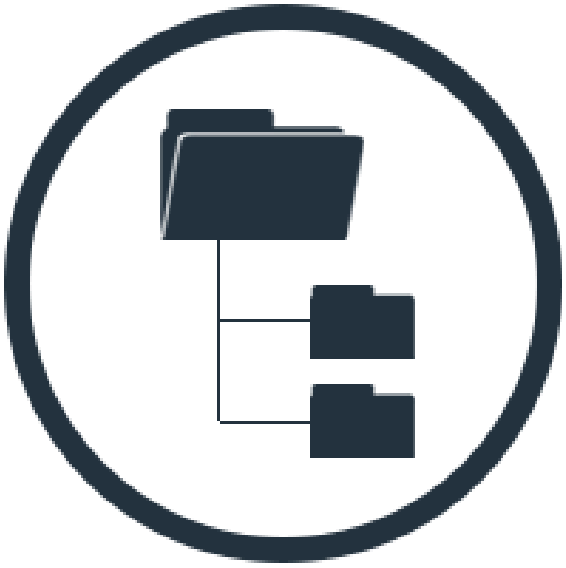
Simple File System Using FUSE

Introduction to Operating System Lab Project

November 2018



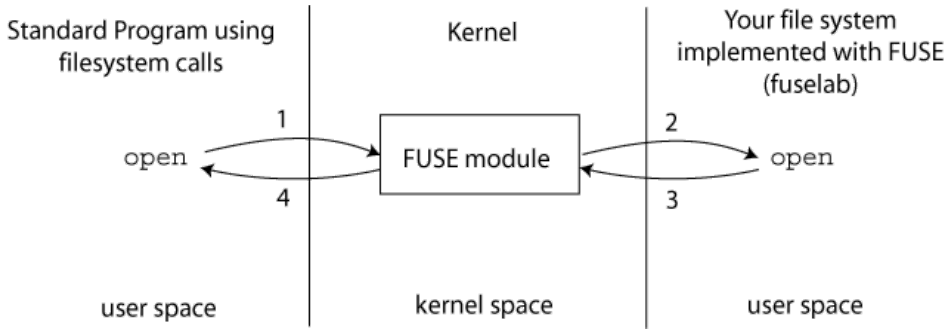
Team:-

Bharath Chandra	01FB16ECS087
B Shashank	01FB16ECS086
Bhavani Shankar	01FB16ECS089

Introduction

Filesystem in Userspace is a software interface that lets non-privileged users create their own filesystem without having to modify kernel code.

FUSE provides an essential set of function prototypes that can be used to implement callbacks for systems calls. These system calls form the backbone of any file system and are called internally when any linux commands are execute



Goals

1. To implement basic function calls that shell commands use to communicate with the file system.
2. To define/modify the underlying stat structure, i.e the structure that is used to store the attributes of each file.
3. To translate the created file structure into data that can be stored in the secondary memory when the file system is unmounted. This data is used to recreate the last state the file system was in.

Implemented Callbacks

The callbacks that were implemented using FUSE are:

- getattr: This function is called when the system command stat is called
- readdir: This function is called when the system command ls is called
- open: This function is called when a file is opened.
- read: This function is called when a file is read.
- utimens: This function is called when an already existing file is accessed through a touch command.
- rmdir: This function is called when rmdir command is used to remove an empty directory. When the removal of a non-empty directory is attempted, an error is displayed.
- mkdir: This function is called when a mkdir command is used to create a new directory.
- create: This function is called when a new file is to be created
- write: This function is used to write to a file.
- truncate: This function is called when an attempt to create an already existing file is made. Then the already existing file is opened after truncating its contents
- unlink : This function is internally called when a file is deleted from a directory. Such a deleted file needs to be ‘unlinked’ from a directory.
- rename: This function is called when the mv command is used to rename a file
- destroy: This function is used to make the file system persistent, and it specifies what is to be done when the file system is unmounted.
- opendir: This function is called when a directory within the file system is opened.

Mounting and Unmounting

Upon completing implementation of the required features, the file system had to
be mounted onto a mount point in the underlying filesystem. This directory would
be considered the new root directory of the created file system.

Requirements

- Hardware
    1. Intel Core i3 2.2 Ghz
    2. 2 GB of RAM
    3. 2 GB Hard Disk Space

- Software
    1. Linux Operating System
    2. Fuse and all its dependencies installed
    3. A folder to mount the File System

To run the code

The following commands were used to test the file system

Compile using : gcc filesys.c -o fs.o `pkg-config fuse --cflags --libs` -w
Mount : ./fs.o <mount directory>

Phase-1 Implementation of Callbacks

- This phase involved the creation of some basic callbacks such as readdir, getattr that are required for the execution of linux commands such as ls, and stat.
- These function callbacks were implemented using the prototypes specified in fuse.h

```
static struct fuse_operations fuse_oper = {
    .getattr    = fsys_getattr,
    .readdir    = fsys_readdir,
    .open       = fsys_open,
    .read       = fsys_read,
    .utimens    = fsys_utime,
    .rmdir      = fsys_rmdir,
    .mkdir      = fsys_mkdir,
    .create     = fsys_create,
    .write      = fsys_write,
    .truncate   = fsys_truncate,
    .unlink     = fsys_unlink,
    .rename     = fsys_rename,
    .destroy    = fsys_destroy,
    .opendir    = fsys_opendir,
};
```

Phase-2 User defined Stat structure

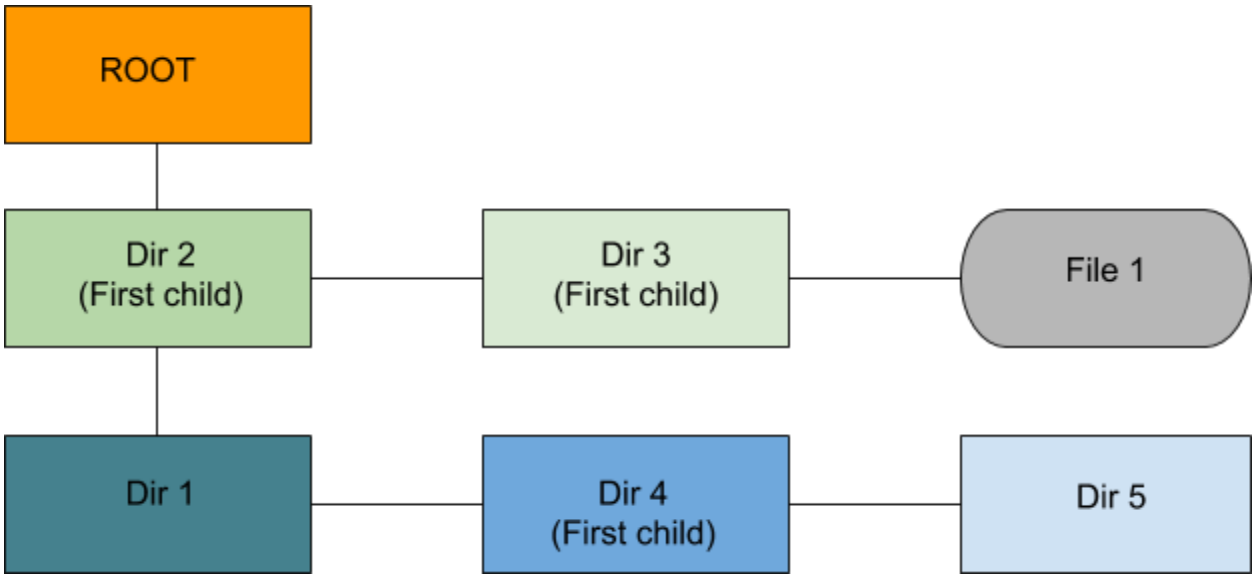
- This phase involved the creation of a user defined stat structure that captured and stored information about a file/directory.
- This stores attributes such as file mode, access/modify/change times, link count etc.
- Some other further callbacks were implemented, such as rmdir, unlink etc.

Phase-3 Persistent Storage

- This phase of the project aimed to establish persistence of the file system
- Files and directories created in the file system remain even when unmounted
- This was done by serialising the contents of the file system and writing it to a binary file.
- This serialised file was created at the same level in the original directory where the implementation file resides.
- Each time the file system was mounted, the contents of the file system are read from the binary file which stores the content of the file if it had previously been mounted.
- If the file system had not been previously been mounted, then the file system is created anew.

Design of the File System

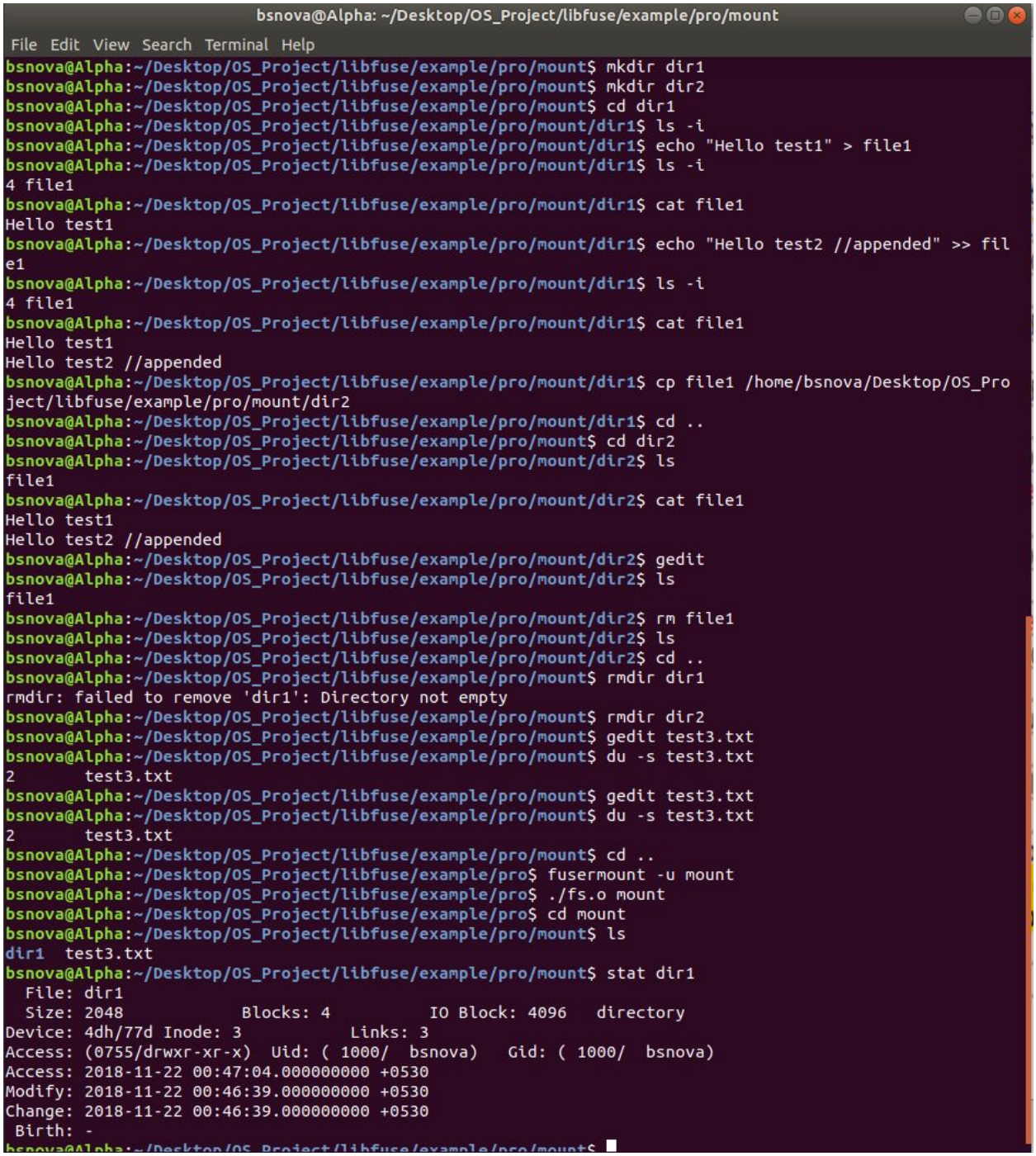
- The file system was implemented as a modified tree structure consisting of hierarchically chained multiple linked lists.
- Every newly created node is placed at the head of the linked list, between the root and the previous first child.
- A node structure was defined where the members of the structure are a pointer to the parent of the node, a pointer to the first child, a pointer to the next node along with the data of the file, and an object of the structure inode.
- The structure inode contains the following members: the name of the node, a flag indicating if the node is a directory or a file, and an object of the user defined stat structure.
- The user defined stat structure contains extensive metadata about the node such as modify time, access time, change time, number of links, and file mode.



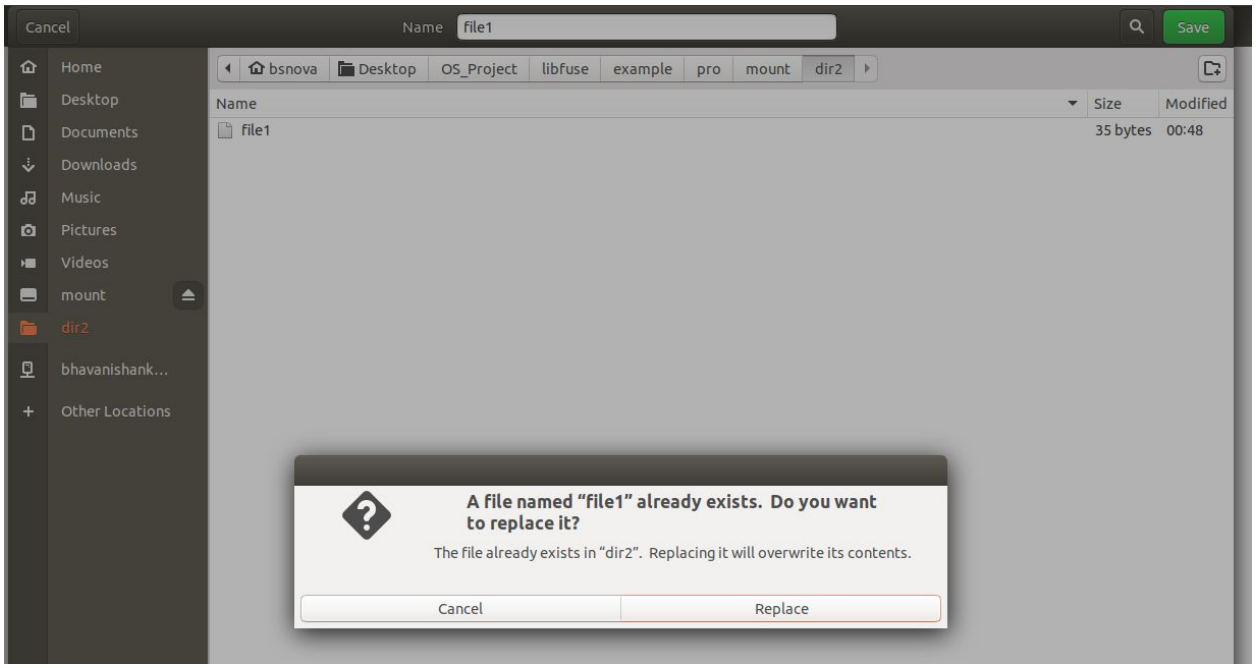
A root directory of size 4096 bytes has been defined. Each block is specified to be of 512 bytes. When a new file is created, a minimum size of 4 blocks is assigned to the file. When the contents of the file exceed this assigned size, the file size if extended by another 4 blocks.

As seen in the above diagram, the root directory has 2 directories under it, and each of those directories has a linked list of files under it. Each newly added file/ directory is added to the head of the linked it.

Test Cases


bsnova@Alpha: ~/Desktop/OS_Project/libfuse/example/pro/mount
File Edit View Search Terminal Help
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount$ mkdir dir1
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount$ mkdir dir2
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount$ cd dir1
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount/dir1$ ls -i
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount/dir1$ echo "Hello test1" > file1
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount/dir1$ ls -i
4 file1
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount/dir1$ cat file1
Hello test1
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount/dir1$ echo "Hello test2 //appended" >> fil
e1
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount/dir1$ ls -i
4 file1
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount/dir1$ cat file1
Hello test1
Hello test2 //appended
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount/dir1$ cp file1 /home/bsnova/Desktop/OS_Pro
ject/libfuse/example/pro/mount/dir2
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount/dir1$ cd ..
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount$ cd dir2
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount/dir2$ ls
file1
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount/dir2$ cat file1
Hello test1
Hello test2 //appended
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount/dir2$ gedit
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount/dir2$ ls
file1
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount/dir2$ rm file1
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount/dir2$ ls
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount/dir2$ cd ..
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount$ rmdir dir1
rmdir: failed to remove 'dir1': Directory not empty
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount$ rmdir dir2
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount$ gedit test3.txt
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount$ du -s test3.txt
2 test3.txt
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount$ gedit test3.txt
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount$ du -s test3.txt
2 test3.txt
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount$ cd ..
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro$ fusermount -u mount
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro$ ./fs.o mount
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro$ cd mount
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount$ ls
dir1 test3.txt
bsnova@Alpha:~/Desktop/OS_Project/libfuse/example/pro/mount$ stat dir1
File: dir1
Size: 2048 Blocks: 4 IO Block: 4096 directory
Device: 4dh/77d Inode: 3 Links: 3
Access: (0755/drwxr-xr-x) Uid: (1000/ bsnova) Gid: (1000/ bsnova)
Access: 2018-11-22 00:47:04.000000000 +0530
Modify: 2018-11-22 00:46:39.000000000 +0530
Change: 2018-11-22 00:46:39.000000000 +0530
Birth: -

Creating a file with same name:



Conclusion

In building a simple file system, the inner workings of file systems were understood at a much finer level. Implementation of a user defined file system allow for many flexibilities, that are not possible with many kernel specific file systems. This flexibilities allow for these user-defined file systems to be application specific, and FUSE simplifies matters by providing prototypes of the main functions (system calls) of a file system.