Simple File System

Design Documentation

Team Members:

- S Jayasurya 01FB16ECS318
- Samarth M. 01FB16ECS335
- Sanat Bhandarkar 01FB16ECS339

Aim:

To build a simple file system using FUSE.

Goal:

To understand the working of a file system by implementing system calls, file I/O and memory management and to familiarize with the concepts of Linux file systems.

File System:

A file system is a **process** that manages how data is stored on a disk and how this data can be accessed and managed.
It has the capability to perform various operations such as opening and closing files, creating and deleting directories and providing file permissions and access methods for a file.

FUSE:

FUSE(File System in User Space) is a framework or a **software interface** for UNIX-like operating systems, that allows users to create their own *virtual* file systems without modifying the kernel code.

FUSE provides various functionalities that make it easier to implement a file system. With FUSE we can mount and unmount the file system and also specifies how the file system is to respond to system call requests. FUSE calls the functions implemented by the user instead of running the Operating System system call code.

Steps to create a Simple File System:

Creating the Inode Structure:

An inode structure contains information about the files and directories in the file system. It also contains information about the entire file system itself. This data about the file system is referred to as the **metadata** (data of data) of the file system.

Our inode structure contains various file parameters such as: file path, name, type, permissions, userid, groupid, number of children, access time, modify time, status change time, creation time, file size and **inode number**.

```
typedef struct inode{
    char * path;
    char * name;
    char * type;
    mode_t permissions;
    uid_t user_id;
    gid_t group_id;
    int num_children;
    time_t a_time;
    time_t m_time;
    time_t c_time;
    time_t b_time;
    off_t size;
    int blk_no[MAX_DIRECT_POINTERS];
    unsigned long int inode_number;
}inode;
```

The inode number is an identifier for each inode entry in the inode table. The above attributes of the file are accessed through its inode number.

Implementing System Calls:

We have implemented some basic system calls that are similar to Linux file systems. FUSE is used to call our functions instead of calling the OS functions.

The system functions implemented include:

1. getattribute
2. readdir
3. open
4. read
5. write
6. mknod
7. mkdir
8. rmdir
9. unlink
10. chmod
11. Chown

At this point, the operations performed on the file system and the files created would be removed from the file system as soon as it was unmounted. Hence, we need persistence.

Making the File System Persistent:

Persistence is essential so that the file system is preserved across multiple machine boots. Persistence was achieved by writing our entire inode structure and datanode structures to a binary file. This binary file is read as soon as the file system is mounted to recover the data. Writes are performed to the binary file whenever a change is made to the inode and datanode structures.
We only read the binary file once, during mount. But multiple writes are performed.

Result

A file system in userspace was built using FUSE and various features of file systems were implemented to understand its working in Operating Systems.