UNIX System Programming

FILESYSTEM IMPLEMENTATION USING FUSE
Non-Persistent and Persistent

Sanketh Rangreji	01FB15ECS267
Saurav Sachidanand	01FB15ECS272
Shahid Ikram	01FB15ECS274

OBJECTIVE

To implement a simple file system in C using libfuse.

The project is broadly divided into three phases:

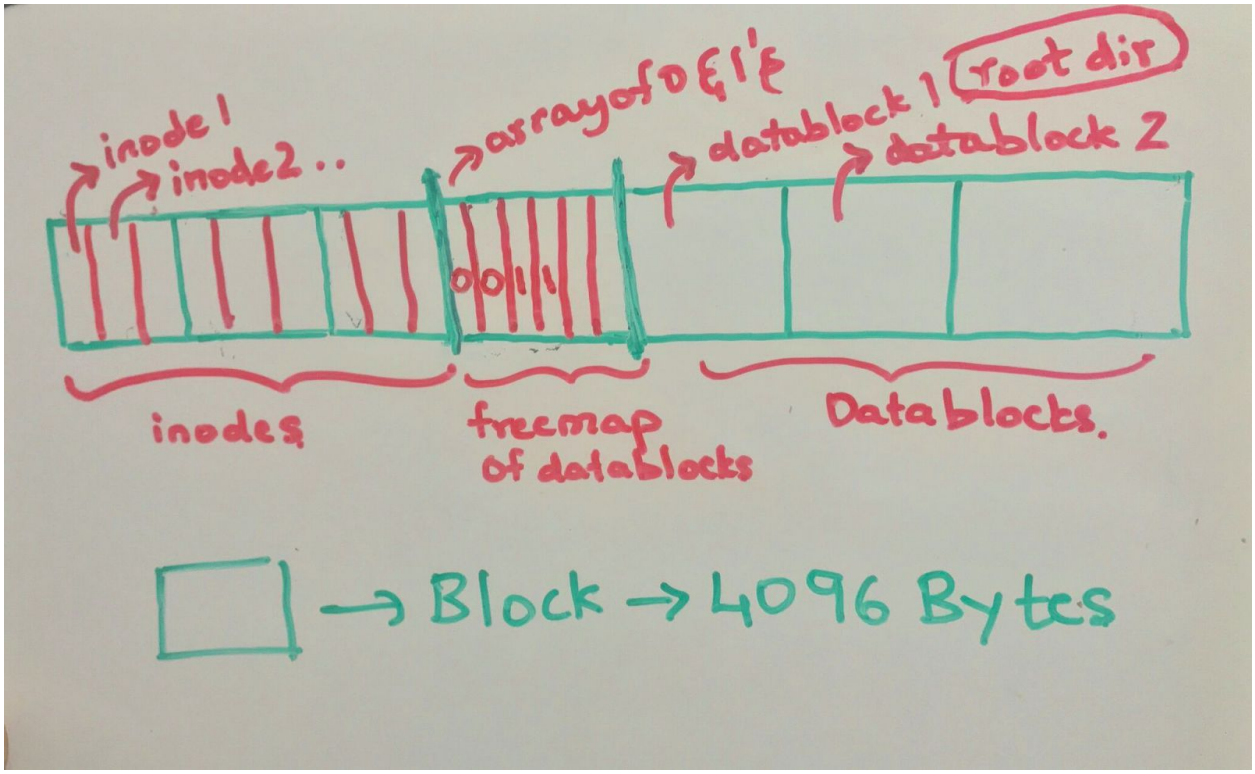
Phase 1 : To set up fuse and experiment with a passthrough file system and implement simple file operations with/ without fuse , without developing file a system structure and allocating blocks in memory.

Phase 2: To come up with a simple file system structure and implement the file system abstractions like the inode blocks, data blocks, directory structure and also to begin porting to secondary memory

Phase 3: To port the file system to secondary memory and ensure persistence of the file system and preserve its state across reboots

THE FILE SYSTEM STRUCTURE

❑ Memory version



Inode Structure for a file

```
typedef struct {
  bool used;            // valid inode or not
  int id;               // ID for the inode
  size_t size;          // Size of the file
  char *data;           // pointer of data block
  bool directory;       // true if its a directory else false
  int last_accessed;    // Last accessed time
  int last_modified;    // Last modified time
  int link_count;       // 2 in case its a directory, 1 if its a file

} __attribute__((packed, aligned(1))) inode;
```

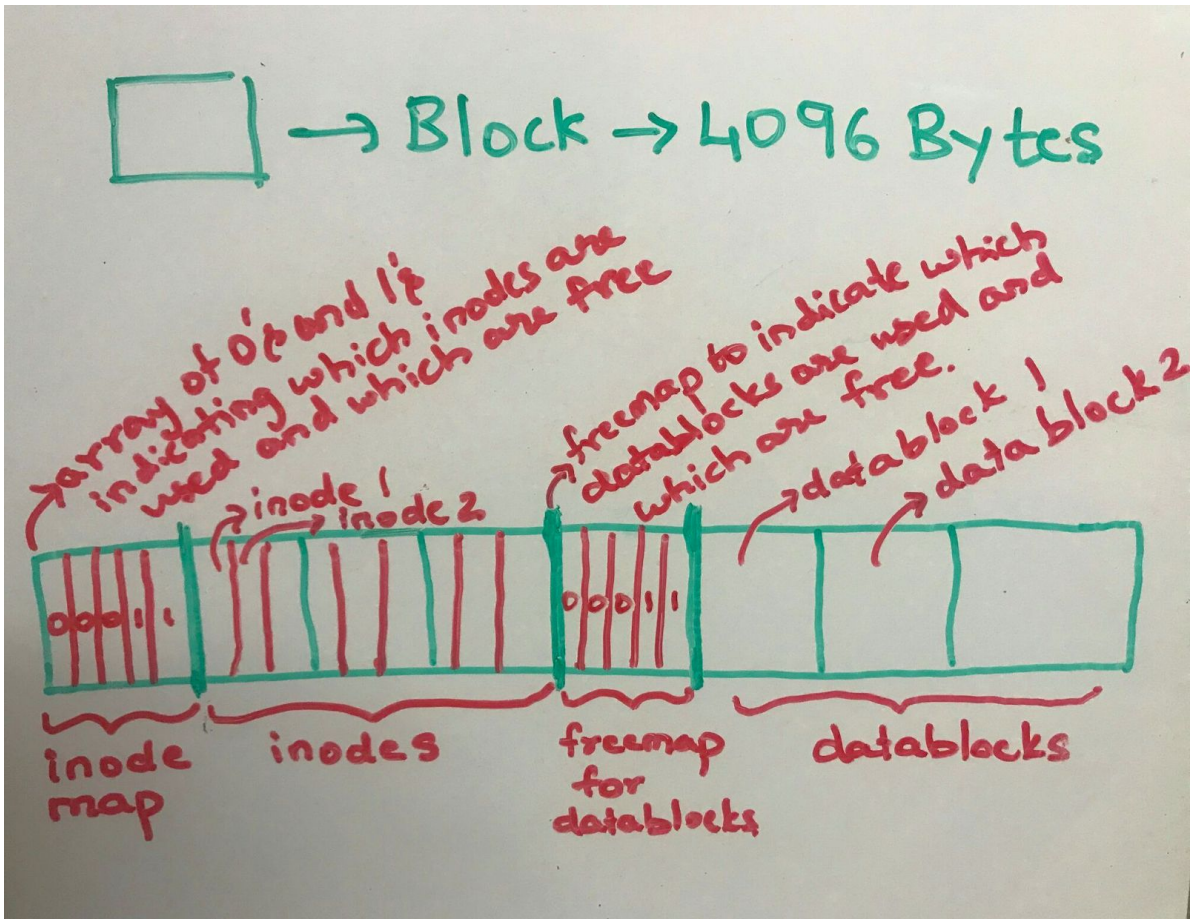
NOTE: address to the datablock is stored in inode

DIRECTORY Entry for a Directory in the datablock

```
typedef struct{
  char *filename;
  inode *file_inode;
}dirent;
```

NOTE: address of the inode for the file is stored.

❑ Disk version



Inode Structure for a file

```
typedef struct {
 bool used;                  // valid inode or not
 int id;           // ID for the inode
 size_t size;        // Size of the file
 int data;         // offset of data block
 bool directory;       // true if its a directory else false
 int last_accessed;      // Last accessed time
 int last_modified;      // Last modified time
 int link_count;       // 2 in case its a directory, 1 if its a file

} __attribute__((packed, aligned(1))) inode;
```

NOTE : Offset of the datablock is stored in the inode not the address for Disk implementation.

DIRECTORY Entry for a Directory in the datablock

```
typedef struct{
 char filename[15];
 int file_inode;
}dirent;
```

NOTE: offset of the inode for the file is stored and not the address of the inode itself

FEATURES OF OUR FILE SYSTEM

```
#define SBLK_SIZE (1 << 4)
#define BLK_SIZE (1 << 12)

#define N_INODES 100
#define DBLKS_PER_INODE 1
#define DBLKS (DBLKS_PER_INODE * N_INODES)

#define ROUND_UP_DIV(x, y) (((x) + (y) - 1) / (y))

#define FREEMAP_BLKS ROUND_UP_DIV(DBLKS, BLK_SIZE)
#define INODE_BLKS ROUND_UP_DIV(N_INODES*sizeof(inode), BLK_SIZE)

#define INODE_MAP_BLKS ROUND_UP_DIV(DBLKS, BLK_SIZE)

#define FS_SIZE (INODE_MAP_BLKS + INODE_BLKS + FREEMAP_BLKS + DBLKS) * BLK_SIZE


#define MAX_NO_OF_OPEN_FILES 10
```

Size of Data Block : 4096 Bytes

Number of Inodes : 100

Number of Data Blocks per Inode : 1

In order to keep the implementation simple, we decided to limit the number of inodes to 100 and to keep only one data block entry per inode. This effectively means that the file system only supports upto a maximum of 100 files and / or directories and each file is limited to 4 KB. The size of the data blocks is arbitrary and can be changed.

As indicated in the file system structure diagram, there is a freemap which is basically a separate block that holds information about which data blocks are free.

The file system always has a hardcoded root directory on start up and all files and directories are nested under this root directory.

Also, The filename of a file or a directory is limited to 15 bytes, ie., 15 characters.

FILES

Each file has an corresponding inode, which holds information about where the file contents are stored, i.e in which data block.

The data block stores character bytes if its a file.

DIRECTORY

All directories are implemented as files. Thus they have a corresponding inode allocated for them, however the data block now stores the directory entry structure and not data.

The directory entry structure holds the name of the file/directory and a pointer/ inode number(memory/disk).

BASIC OPERATIONS SUPPORTED

- ❏ ls command
- ❏ cat command
- ❏ rm command
- ❏ rmdir command
- ❏ echo command (to redirect to file and write to file)
- ❏ cp command
- ❏ Appending to existing file

```
static struct fuse_operations fs_oper = {
 .init       = fs_init,
 .getattr    = fs_getattr,
 .readdir    = fs_readdir,
 .mkdir      = fs_mkdir,
 .rmdir      = fs_rmdir,
 .open       = fs_open,
 .create     = fs_create,
 .read       = fs_read,
 .write      = fs_write,
 .unlink     = fs_rm,
};
```

PERSISTENCE OF THE FILE SYSTEM

For achieving persistence , the entire file system allocated in memory was written to a file.

During initialization of the file system, if there wasn't any previously existing file system to restore, then a new file system is created, else the file(disk) that stores the file system structure is read into a memory block and further operations are performed.

Whenever any change is made to the filesystem such as: Creation of a file, Writing into a file, Appending to a file, creation of a directory, deletion of a directory and deletion of a file the update made in memory is immediately written into the disk.

In case of other operations such as reading a file etc, no change is made and hence the disk isn't updated, to ensure efficiency.