TurboFS

A FUSE file system based on EXT2

TEAM
MANISH SHETTY M : 01FB16ECS192
MEGHA KALAL : 01FB16ECS203
MEGHANA G : 01FB16ECS205
MIRIAM JOHN K : 01FB16ECS209

OBJECTIVE

To implement a simple file system in C using libfuse. The project is broadly divided into three phases:

1. Phase 1 : To set up fuse and implement simple file operations with/without fuse , without developing file a system structure and allocating blocks in memory.
2. Phase 2: To come up with a simple file system structure and implement the file system abstractions:
    a. inode blocks,
    b. data blocks,
    c. directory structure
    d. bitmaps
3. Phase 3: Port the file system to secondary memory and ensure persistence of the file system and preserve its state across mounts .

SPECIFICATIONS

- SuperBlock Size : 2^4 = 16 bytes
- Block Size : 2^13 = 4096 bytes
- No_Of_Inodes = 100
    - Max_No_Of_Files = 100
    - Also to make the complexity lesser we limited the size of 1 file to 1 datablock.
    - **Data Blocks = 100 BLOCKS**

- SizeOf(Inode) = 30 bytes
    - Thus total memory for inodes = 30 x 100 = 3000 bytes
    - 3000 < 4096 ⇒ **Inodes = 1 BLOCK**

- Inode Bitmap : An array of 1/0 (integer = 4 bytes) for each inode
    - No of Inodes = 100
    - Memory required = 100 x 4 = 400 bytes < 4096
    - **Inode Bitmap = 1 BLOCK**

- Similarly **Data Bitmap = 1 BLOCK**

- The file system has has a hard coded **root directory** dirent pointer(global) which makes any access easier and quicker.
    - All other files and directories are nested under root

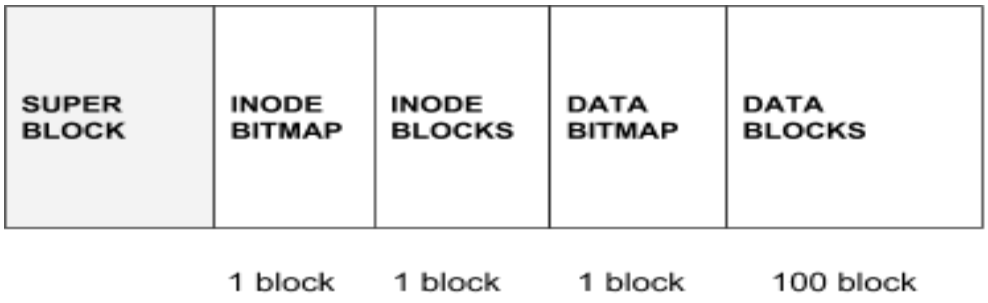
- The filename/ directory name is limited to 15 bytes (ie: 15 characters)

Therefore
total no of blocks = 100 + 1 + 1 + 1 = **103 blocks**

TOTAL MEMORY of File System
= 103 Block x 4096 bytes per block
= 42188 bytes
= 42.18 Kb

DESIGN

This is a representation of how the file system is stored in the memory.



The file system has the following:

- **Super Block** - contains metadata for the file system.
- **Inode Bitmap** - an array which represents if the corresponding inode is free or used.
- **Inode Blocks** - store the actual inode structure .
- **Data Bitmap** - similar to inode bitmap except that it indicates if a data block is used or free.
- **Data Blocks** - here is where the actual file or directory is stored. The first data block contains the root directory.

Super Block Structure

```
typedef struct
{
 int magic;
 size_t blocks;
 size_t iblocks;
 size_t inodes;
} __attribute__((packed, aligned(1))) sblock;
```

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 data block is stored for a file in the inode , not the address so that Disk storage and retrieval becomes easier.

FILES AND DIRECTORIES:

- Files and directories are stored in the data blocks.
- Files are stored as character bytes in the data blocks.
- Directory information is stored in the data blocks as dirent structure. The dirent structure stores the file names of files in the directory with their corresponding inode number.

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

// Size of dirent = 16 bytes
// Hence in a 4KB block we can have 256 directory entries
```

NOTE : Offset of the inode in the inode block is stored for a file in the dirent , not the address of the inode itself.

BASIC OPERATIONS SUPPORTED:

- ➢ls
- ➢cd
- ➢cat
- ➢rm
- ➢echo
- ➢touch
- ➢mkdir
- ➢rmdir
- ➢cp

These operations are all supported by the working of fuse and it's redirection of system calls to TurboFS function calls. The implemented functions are :

```
static struct fuse_operations turbo_oper =
{

  .getattr    = turbo_getattr,
  .readdir    = turbo_readdir,
  .mkdir      = turbo_mkdir,
  .rmdir      = turbo_rmdir,
  .open       = turbo_open,
  .create     = turbo_create,
  .read       = turbo_read,
  .write      = turbo_write,
  .unlink     = turbo_rm,

};
```

PERSISTENCE OF THE FILE SYSTEM:

To provide persistence the file system needed to be written to the disk. To do this the entire file system is written into a binary file which acts as a disk. So essentially the entire file system is placed in a single file on disk.

Initially as the file system is not on disk it must be created. Subsequently whenever the file system is accessed the data must be read from the file(disk) onto the memory. This read is a one time step as soon as we mount the FS onto memory.

Further changes made in the file system will be reflected on the disk and will persist . To make the writing process optimal we used a simple **SERIALIZATION** technique which allows us to write any data structures into a file as binary.