CHARMinar

A Linux-Based File System using FUSE

Harshavardhan Miryala

Taanya Nithya Anand

Tim Kim

Brian Lee

Introduction

- EXT model for implementing the file system
- Implemented in C
- No non-standard library / package used
- Test suites in C / Python
- Broke it to four layers (hierarchical)
  - Disk Layer
  - Block Layer
  - File Layer

Disk Layer

- Everything is communicated only in terms of blocks
- Block Size - 4096
- File System Size - 30 GB
- Takes cares of any data manipulation at the block level

Block Layer

- Three segments
  - Super block (God and needs to be created first)
  - Inodes Blocks (each block stores multiple inodes)
  - Data Blocks (traversed with the help of free-list head)
- 10% of blocks allocated to inodes
- Super block stores information about latest free inum and free list head

Inode

- 10 direct blocks
- 1 each single, double and triple indirection
- Max file that can be stored ~ 500 GB (if storage space is available :P)
- We have tested creating 8 GB file in a partition of size 10 GB
- Inodes per block - 21
- We use latest_inum to figure the next free inum for inode allocation - average case O(1), worst case - O(N)

Data blocks

- Free list block and normal blocks
- One free list block is linked to the other free list block
- Each free list block has information about availability of (BLOCK_SIZE/ADDRESS) data blocks, i.e. 512
- Initially has data block num of all blocks tracked and the link to next free list block
- We make a value at an index to 0 to indicate occupancy
- During freeing, add back the dblock num to free list head

File Layer

- Directories are also looked up as files
- File has fblock num which makes it easier to locate the dblock num (direct block / single / ….)
- Files info inside a directory are stored in the data block which has the information about inode_num, special num, file name length and file name.
- Special Num is important to deal with flexible file name lengths and avoid dealing with re-arranging the files' info once a file is deleted.

Contd...

- Main functionalities
  - Adding or removing files in a dir
  - Read / Write from a file
  - unlink
  - mkdir and rmdir
  - rm -rf takes care of clearing inum/s, all data block/s used
  - Path name resolution

LRU Cache

- For quick iname resolution i.e. path name to inode, LRU cache is implemented (instead of looking at the file in parent directory)
- Advantages
  - Fixed size
  - Least used iname will be removed
  - Showed better performance compared to hash table
  - Addition, Updation, Removal - O(1)

Fuse Layer

```
static const struct fuse_operations fuse_ops = {
    .access   = charm_access,
    .chown    = charm_chown,
    .chmod    = charm_chmod,
    .create   = charm_create,
    .getattr  = charm_getattr,
    .mkdir    = charm_mkdir,
    .rmdir    = charm_rmdir,
    .unlink   = charm_unlink,
    .truncate = charm_truncate,
    .mknod    = charm_mknod,
    .readdir  = charm_readdir,
    .open     = charm_open,
    .read     = charm_read,
    .write    = charm_write,
    .utimens  = charm_utimens,
    .rename   = charm_rename,
};
```

Testing

- White box testing
  - Flow of coding - Header, Test Cases, Source Coding, Testing
  - Each layer
  - file / directory creations using scripts covering race conditions during inode creation and data block allocation such as nested directories, multiple files / directories in a directory
  - lseek and unlink
  - Large file creation - 8 GB
- Black box testing
  - Vim
  - GCC
  - Numpy, Pandas

Learnings

- Importance of unit testing and block testing and how it made life easy
- Importance of writing robust code, and it helped us in multiple cases
  - Minimal time to onboard triple indirect
  - Minimal time to create new functionalities which only use existing ones with no complex logic around it
- Clear understanding of file system and the scope for improvements and quite amazed by seeing things work

Improvements

- Bit masking for inode / free list allocation
- Didn’t implement Symlink (due to time constraints)
- Hacky mv op (copy old to new, delete old)

Thanks !!!