CSCI4730/6730 – Operating Systems

Project #4

Due date: 11:59pm, 05/06/2022

Description

In this project, you will implement a simple EXT-like file system simulator to understand the hierarchical directory and inode structures. You will be able to (1) browse the disk information, file and directory list, (2) create and delete files and directories, (3) read files. The functionality of the file system is similar to EXT file systems, but it does not include per-process open file table, permission and user management.

Figure 1 shows the file system structure of the simulator. A disk block size is 512 byte and the disk has 4096 blocks. Superblock, inode map, block map, and inode blocks are loaded into the memory at file system mount time. A size of inode entry is 128 byte, and 4 inode entries are stored in a single disk block.

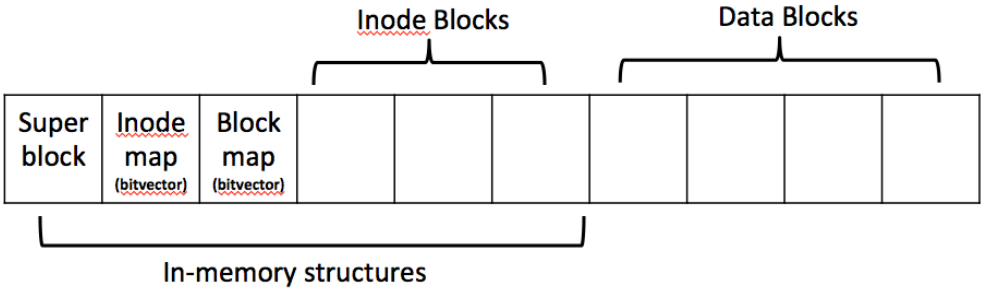

Fig 1. File System Structure

You can start the simulator with the following command:

$./fs_sim <disk name>

It has one command-line argument, <disk name> to specify the file name that stores our file system. If <disk_name> is not exist, the simulator creates a new file system and mounts it. If <disk_name> is already exist, the simulator mounts it.

After the simulator mounts the file system, it waits for the commands from the user.

It supports commands to access files and directories, and “quit” or “exit” to unmount the file system.

You can also feed commands directly from the input file as follows:

$./fs_sim test.disk < testfile.input

You may use "fs_sim_reference" executable to generate reference output and compare it with yours. Note that it is not yet fully tested and could have bugs and can crash.

Part 1: File – 50%

In this file system, the super block is stored in the disk block 0. It contains the number of available inode and data blocks, and root directory information. Super block is defined in "fs.h" as follows:

```
typedef struct {
  int magicNumber;
  int freeBlockCount; // # of available data blocks
  int freeInodeCount; // # of available inodes
  int rootDirInode;   // root directory
  char padding[496];  // padding bytes to make superblock to 512 bytes
} SuperBlock;
```

In this project, each inode uses 15 direct blocks and 1 single-indirect block to support files up to 73,216 (7,680 +65,536) bytes. Specifically, 15 direct blocks support 7,680 bytes (15 * 512 bytes) and 1 indirect block supports 73,216 bytes (128 * 512 bytes). Each file in a file system has an identification number, called inode number, which is unique in a file system. Inode structure (Inode) is defined in "fs.h" file as follows:

```
typedef struct {
  TYPE type;  // file or directory
  int size;   // file size
  int blockCount;  // # of blocks used
  int directBlock[15];
  int indirectBlock;
  int linkCount; // for hardlink
  char padding[48]; // Padding bytes to make the inode size to 128 byte
} Inode; // 128 byte
```

The file system will support the following functionalities (The current file system already supports “df”, “create”, and “stat”. You will need to implement “cat”, “rm”, “ln”, and “cp”) .

Command	Arguments	Description
df		Show the file system information: number of free blocks / number of free inodes.
stat	<name>	Show the status of the file or directory with name <name>. It displays the inode information; whether it is a file or directory; size of the file/directory; number of blocks allocated; other information stored about this file/directory.
create	<name> <size>	Create a file with <filename> as a name in the current directory, and fill it with <size> amount of random characters.
cat	<name>	Print out the content of the file.
cp	<src> <dest>	Copy <src>file to <dest>. Both <src> and <dest> are in the current directory.
rm	<name>	Delete <name> file in the current directory.
ln	<src> <dest>	Create a hard link. Both <src> and <dest> are in the current directory.

You will need to modify “**file.c**” file to implement Part 1. You can use pre-defined APIs to access inode, inode map, block map, and disk blocks. All APIs are declared in “API.h” file.

```
// Get one available inode from inode map.
// It returns the first available inode number, and decreases freeInodeCount in the SuperBlock.
int get_inode();

// It frees a inode and increase freeInodeCount in the SuperBlock.
void free_inode(int inodeNum);

// Get one available data block from block map.
// It returns the first available block number, and decreases freeBlockCount in the SuperBlock.
int get_block();

// It frees a block and increase freeBlockCount in the SuperBlock.
void free_block(int blockNum);


// Return a inode structure.
Inode read_inode(int inodeNum);
// Write a inode structure to inode blocks.
void write_inode(int inodeNum, Inode newInode);


// Read a data block (512bytes) from the disk.
void read_disk_block(int blockNum, char *buf);
// Write a data block to the disk.
void write_disk_block(int blockNum, char *buf);

// Search "name" from the current directory and return inode number, if exist.
// It returns -1 if "name" is not exist in the current directory.
int search_cur_dir(char *name);
```

Part 2: Directory – 50%

In this part, you will implement hierarchical, tree-structured directory. In our simulator, each directory has up to 25 entries (files and sub-directories).

The first entry is always a current directory (“.”) and the second entry is a parent directory(“..”). The root directory does not have a parent directory. You will need to implement the following functionalities (you will need to implement “mkdir”, “rmdir”, and “chdir”):

Command	Arguments	Description
ls		Show the content of the current directory
mkdir	<name>	Create a sub-directory <name> under the current directory.
rmdir	<name>	Remove the sub-directory <name>. It only removes an empty directory. “.” and “..” cannot be deleted.
cd	<name>	Change the current directory to <name>

A directory structure (Dentry) is 512 bytes, and it is stored in a single disk block. Directory structure is defined in “fs.h” file as follows:

```
typedef struct {
  char name[MAX_FILE_NAME]; // max length of file/directory name is 20 bytes.
  int inode;
} DirectoryEntry;

typedef struct {
  int numEntry;  // number of files and sub-directories
  DirectoryEntry dentry[MAX_DIR_ENTRY]; // each entry has "name" and "inode"
  char padding[4]; // padding bytes to make it to 512 byte
} Dentry;
```

To create a new directory, you will need one inode and one data block to store the directory structure (Dentry).

The filesystem always maintains “curDir” that contains the directory structure of the current directory. When the user changes the current directory using “cd”, you need to properly change “curDir”.

```
// directory structure and inode for the current directory.
// curDir always points to the current directory.
// curDirInode is inode number for the current directory.
// It should be properly changed if the user changes the directory by "cd" command.
extern Dentry curDir;
extern int curDirInode;
```

You will need to modify “**directory.c**” file to implement Part 2.

Submission

Submit a tarball file through eLC with all project files include:

1. README file with:
    a. Your name
    b. List what you have done and how to test them. So that you can be sure to receive credit for the parts you've done.
    c. Explain your design.
2. file.c and directory.c files. If you made further modification, make sure to include all modified files.
    a. Makefile
    b. All modified source and header files
    c. Do not submit object or executable files
3. **Your code should be compiled in odin machine. Any compilation error on Odin will lead to 0 point.** Make sure to use Makefile and command "make" to compile the project on Odin. If you have any special compilation needs, make sure to modify Makefile accordingly.
4. Submit a tarball through ELC.