CSE 312/504 – Operating Systems - Spring 2020

Midterm Exam Project Report

Student Name: Ozan Şelte

Student Number: 161044061

The filesystem is represented as 1 MB sized *uint8_t* array.

General Structure:

144 B **Superblock**	*1 048 432 B* *Left Space*

In the beginning, spaces will be calculated. According to the free inode count, the maker program easily finds the block count. Blocks align to the rightmost and inodes place just left of them. Between the superblock and inodes, there will occur a blank space, padding.

Suppose that; blocks are 4KB, free inode count is 400(with "/" there will be a total 401 inodes) and inode size is 32 B. The example table is shown like this:

144 B **Superblock**	*3 408 B* *Empty*	**32 B** **Inode 0**	12 768 B Inodes 1-399	**32 B** **Inode 400**	**4 096 B** **Block 0**	1 024 000 B Blocks 1-250	**4 096 B** **Block 251**

Supernode Struture:

```
struct SuperBlock {
    uint16_t blockSize;
    uint16_t blocksCount;
    uint16_t inodesCount;
    uint16_t filesCount;
    uint32_t firstBlockAddr;
    uint32_t firstInodeAddr;
    uint8_t blocksBitmap[128];
} __attribute__ ((__packed__));
```

blockSize: Block size as KBs. (4)

blocksCount: Block count. (252)

inodesCount: Inode count. (401)

filesCount: File count in a single block, (blockSize*KB)/(FileEntrySize). (128)

firstBlockAddr: First block's offset according to the beginning. (16384)

firstInodeAddr: First inode's offset according to the beginning. (3552)

blockBitmap: Bitmap for free blocks.

The superblock has a fixed length as 144 B. Every filesystem that created by the maker program begins with the superblock. The values given after the explanations belong to the given example table above.

I-Node Structure:

```
struct Inode {
    uint8_t linkCount;
    uint8_t type;
    uint32_t size;
    int32_t lastTime;
    uint16_t direct[DRCT_CNT];
    uint16_t singleI[IND1_CNT];
    uint16_t doubleI[IND2_CNT];
    uint16_t tripleI[IND3_CNT];
} __attribute__ ((__packed__));
```

```
#define DRCT_CNT (8)
#define IND1_CNT (1)
#define IND2_CNT (1)
#define IND3_CNT (1)
#define ADDR_SIZE (sizeof(u
#define KEY_LEN (30)
#define T_FIL (0)
#define T_DIR (1)
#define T_SYM (2)
```

linkCount	Link count to the Inode. If equals to 0, that means the inode is free, not using.
type	Type of the Inode. *T_FIL* = file, *T_DIR* = directory, *T_SYM* = symbolic link
size	Size of the content as Bytes.
lastTime	Last modification type, 32 bit *time_t*.
direct	Addresses of direct block entries, length can change with *DRCT_CNT*.
singleI	Addresses of single indirect block entries, length can change with *IND1_CNT*.
doubleI	Addresses of double indirect block entries, length can change with *IND2_CNT*.
tripleI	Addresses of triple indirect block entries, length can change with *IND3_CNT*.

An Inode contains this information about a file. Directories begin with 2*sizeof(struct File), these belong to “.” and “..” directions.

Block entry counts can change from 0, this feature is very useful to test indirect block entries. DRCT_CNT cannot be 0 because it is using for symbolic links and directories for the first two dotfiles. When DRCT_CNT=1, IND1_CNT=0, IND2_CNT=0, IND3_CNT=1; we can easily test triple indirect blocks.

T_SYM files obtain file path in their first block(direct[0]) so their size will be the path length.

Directory/File Entry Structure:

```
struct File {
    uint16_t inode;
    char name[KEY_LEN];
} __attribute__ ((__packed__));
```

If an Inode’s type is T_DIR it contains size/sizeof(struct File) file entries. The first two of them are dotfiles. With this method, we can quickly calculate the count of files in a directory. KEY_LEN is 30, so a filename can be at most 29 characters long, and every File entry has a constant 32 bytes size. Every file entry points to an inode and for every pointing, that inode’s linkCount increases. When del or rmdir called, linkCount decreases, if equals to 0, it is deleting.

Free Blocks and I-Nodes:

Inodes have linkCount, if it is 0, Inode is free. Blocks have a bitmap in the superblock.

All Functions:

struct Inode *getInode(uint8_t *fs, uint16_t num);
Returns the inode which at the given position.
uint8_t *getBlock(uint8_t *fs, uint16_t num);
Returns the block which at the the given position.
uint16_t firstEmptyBlock(uint8_t *fs);
Returns first empty/free block's number.
uint16_t firstEmptyInode(uint8_t *fs);
Returns first empty/free inode's number.
struct File *getFileInDir(uint8_t *fs, struct Inode *inode, uint64_t num);
Returns file entry at the given position in the directory.
struct File *firstEmptySlot(uint8_t *fs, struct Inode *inode);
Returns first unused file entry in the directory. Initializes new block(direct/indirect) if necessary.
struct File *lastFullSlot(uint8_t *fs, struct Inode *inode);
Returns last using file entry in the directory.
uint8_t *emptyFileBlock(uint8_t *fs, struct Inode *inode);
Returns first unused file block in the file. Initializes new block(direct/indirect) if necessary.
uint8_t *fileBlock(uint8_t *fs, struct Inode *inode, uint64_t num);
Returns file block at the given position in the file.
uint8_t fileBlockNum(uint8_t *fs, struct Inode *inode, uint64_t num, uint16_t *res);
Returns the using blocks(*res*) and count, at the given position in the file. If *num=9*, (2, [x, y]) returns.
void findUpperPath(char *dest, char *str);
Copies upper path to *dest*. If *(“”, “/ysa/usr/file1”)* given, it changes (“/ysa/usr”, “/ysa/usr/file1”).
struct File *getFile(uint8_t *fs, struct Inode *inode, char *name);
Returns given file name's entry in the given directory.
struct Inode *findPathInode(uint8_t *fs, char *path);
Returns given path's I-Node.
size_t singleLength(char *path);
Returns given path's step lenght. If *(“ysa/user/direcotryABC/x”)* given, it returns 3.
void freeFileBlocks(uint8_t *fs, struct File *file);
Frees given file's blocks. Works on direct or indirect blocks.
uint16_t emptyBlocksCount(uint8_t *fs);
Returns count of the empty blocks on the filesystem.
uint16_t emptyInodesCount(uint8_t *fs);
Returns count of the empty I-Nodes on the filesystem.
void checkCapacities(uint8_t *fs, uint8_t checkBlocks, uint8_t checkInodes);
Checks the filesystem has enough blocks and I-Nodes or not. If not it prints error.
void traverseDirs(uint8_t *fs, uint8_t *vMap, uint8_t *counts, struct Inode *inode);
Recursively traverses all the directories from (root) and fills *counts* array. *fsck* is using this function. *vMap* is the bitmap to avoid reprinting.
void printInfos(uint8_t *fs, uint16_t *counts, uint8_t *vMap, struct Inode *inode);
Recursively traverses all the files. Calls *printInode* for every I-Node just for once. *dumpe2fs* uses this to print inode informations. *vMap* is the bitmap to avoid reprinting.
void printInode(uint8_t *fs, uint16_t num);
Prints I-Node information for *dumpe2fs*. Find all links and prints names with block numbers.
void listCmd(uint8_t *fs, char *path);
list command. Prints files in the path.

void mkdirCmd(uint8_t *fs, char *path);
mkdir command. Creates a new directory in the path. New directory will have 2 file entries for dots.
void rmdirCmd(uint8_t *fs, char *path);
rmdir command. Removes the directory on the path if it is empty. If I-Node's *linkCount* equals 0, calls *freeFileBlocks*.
void dumpe2fsCmd(uint8_t *fs);
dumpe2fs command. Prints expected output according to the project pdf.
void writeCmd(uint8_t *fs, char *path, char *linuxFile);
write command. Finds the path; if exists remove the data, if not creates a new file, writes the Linux file's data to inner file.
void readCmd(uint8_t *fs, char *path, char *linuxFile);
read command. Writes to the Linux file, the inner file's data.
void delCmd(uint8_t *fs, char *path);
del command. Finds the path. If it is a file or a symbolix link, deletes it. If I-Node's *linkCount* equals 0, calls *freeFileBlocks*.
void lnCmd(uint8_t *fs, char *filePath, char *linkPath);
ln command. Creates a new file entry in the path, point to the other files I-Node, increases I-Node's *linkCount*.
void lnsymCmd(uint8_t *fs, char *filePath, char *linkPath);
lnsym command. Creates a new file entry, I-Node and block. Block contains the path that linked.
void fsckCmd(uint8_t *fs);
fsck command. Prints expected output according to the project pdf.