GTU Department of Computer Engineering
CSE 312 - Spring 2023
Homework 3 Report

Emirkan Burak Yılmaz
1901042659

Contents

1 PART1

1.1 Structures

The file system implementation includes several structures that represent different components of the file system.

1.1.1 Superblock

The "superblock" structure represents the superblock of the file system. It contains information about the file system configuration, such as block size, positions of important elements (root directory, FAT, data blocks), and the number of blocks in the file system.

```
struct superblock {
    int block_size;
    int root_directory_position;
    int fat_position;
    int data_block_position;
    int num_blocks;
};
```

1.1.2 FAT Entry

The "fat_entry" structure represents an entry in the File Allocation Table (FAT). The FAT is used to keep track of the allocation status of data clusters. Each FAT entry holds a value that indicates the state of a cluster, such as free, bad, or end of file.

```
/**
 * 0x000: Indicates that the cluster is free and available for allocation.
 * 0xFF8: Represents a bad cluster that should not be used.
 * 0xFF7: Marks the end of a file or directory chain.
*/
struct fat_entry {
    unsigned short value;
};
```

1.1.3 Disk Block

The "disk_block" structure represents a disk block in the file system. It includes a reference to the next disk block in a chain and a data buffer to store file or directory data.

```
struct disk_block {
    unsigned short next_disk_block;
    char *data;
};
```

1.1.4 Directory Entry

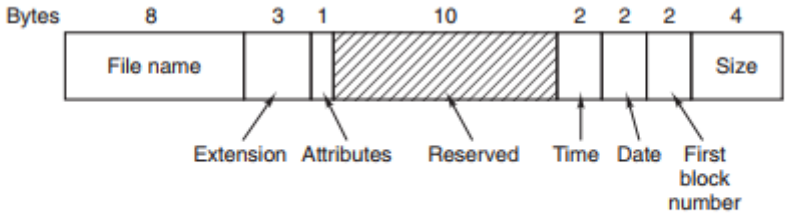
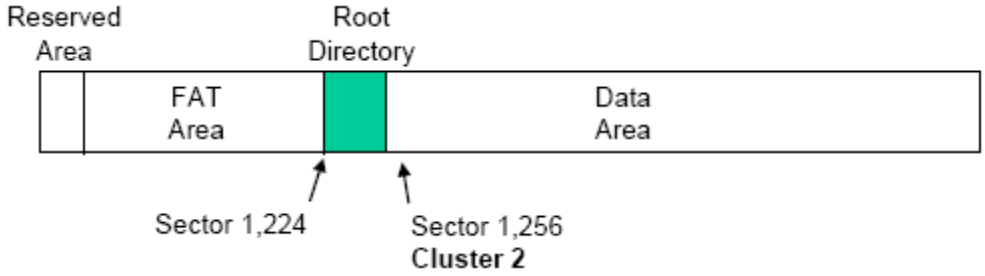


Figure 4-30. The MS-DOS directory entry.

The "directory_entry" structure represents an entry in the file system's directory. It contains information about a file or directory, including the filename, attributes, file size, last modified date and time, and the starting block of the file's data.

```
struct directory_entry {
    char filename[MAX_FILENAME_LENGTH];
    char attributes;
    unsigned short file_size;
    unsigned short last_modified_date;
    unsigned short last_modified_time;
    unsigned short starting_block;
};
```

1.1.5 File System



The "filesystem" structure represents the entire file system. It encapsulates the superblock, FAT entries, directory entries, and disk blocks.

```
struct filesystem {
    struct superblock sb;
    struct fat_entry fat[NUM_ENTRY];
    struct directory_entry directory[NUM_ENTRY];
    struct disk_block dbs[NUM_ENTRY];
};
```

1.2 Functions:

The file system implementation provides various functions to perform operations on the file system and manipulate files and directories.

1.2.1 fs_create(int block_size):

This function creates a new file system with the specified block size. It initializes the superblock, FAT entries, directory entries, and disk blocks. It returns a pointer to the created file system structure.

1.2.2 fs_sync(const struct filesystem *fs, const char *path):

The "fs_sync" function writes the file system's data to a file specified by the provided path. It saves the superblock, FAT entries, directory entries, and disk blocks to the file.

1.2.3 fs_mount(struct filesystem *fs, const char *path):

The "fs_mount" function reads the file system's data from a file specified by the provided path. It loads the superblock, FAT entries, directory entries, and disk blocks from the file into the file system structure.

1.2.4 fs_free(struct filesystem *fs):

This function releases the memory allocated for the file system structure and its associated components, including disk blocks.

1.2.5 fs_print(const struct filesystem *fs):

The "fs_print" function prints the information about the file system. It displays details such as the superblock information, FAT entries, and directory entries.

1.2.6 create_file(struct filesystem *fs, const char *filename):

The "create_file" function creates a new file in the file system. It checks if the file already exists, finds a free directory entry, allocates consecutive free clusters in the FAT for the file, and updates the directory entry with the file's attributes and starting cluster.

1.2.7 delete_file(struct filesystem *fs, const char *filename):

The "delete_file" function deletes a file from the file system. It finds the directory entry corresponding to the file, frees the clusters allocated to the file in the FAT, and marks the directory entry as unused.

1.2.8 read_file(struct filesystem *fs, const char *filename, char *buffer, int size):

The "read_file" function reads data from a file in the file system. It finds the directory entry corresponding to the file, reads the file data from the clusters allocated to the file using the FAT, and copies the data into the provided buffer.

1.2.9 write_file(struct filesystem *fs, const char *filename, const char *data, int size):

The "write_file" function writes data to a file in the file system. It finds the directory entry corresponding to the file, writes the data to the clusters allocated to the file using the FAT, and updates the file size and modification time in the directory entry.

1.2.10 list_directory(struct filesystem *fs):

The "list_directory" function lists the files and directories in the file system. It iterates through the directory entries, retrieves their attributes, and prints the file names and other details.

2 PART2

2.1 makeFileSystem [blocksize] [disk file name]

```c
int main(int argc, char *argv[])
{
    struct filesystem *fs;
    int rv, blocksize;
    char *diskpath;

    if (check_args(argc, argv, &blocksize) == -1) {
        fprintf(stderr, "usage: %s [blocksize] [disk filename]\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    diskpath = argv[2];

    /* Convert to block size from kilobyte to byte */
    blocksize *= 1024;

    fs = fs_create(blocksize);
    if (fs == NULL) {
        fprintf(stderr, "Failed to create file system\n");
        exit(EXIT_FAILURE);
    }

    /* Sync the diskfile with the filesystem object */
    rv = fs_sync(fs, diskpath);
    if (rv == -1) {
        perror("fs_sync");
        exit(EXIT_FAILURE);
    }

    /* Mounting from the disk file */
    rv = fs_mount(fs, diskpath);
    if (rv == -1) {
        perror("fs_sync");
        exit(EXIT_FAILURE);
    }

    fs_print(fs);

    fs_free(fs);
}
```

The code takes command line arguments for the block size and disk filename, creates a file system with the specified block size, synchronizes the file system with the disk, and prints the file system information.

2.2 Running and Results

```
ebylmz@ebylmz: ~/cse/Operating-Systems/hw/hw03/src
ebylmz@ebylmz:~/cse/Operating-Systems/hw/hw03/src$ make clean
rm -rf makeFileSystem mySystem.dat
ebylmz@ebylmz:~/cse/Operating-Systems/hw/hw03/src$ l
ebylmz@ebylmz:~/cse/Operating-Systems/hw/hw03/src$ ls -l mySystem.dat
ls: cannot access 'mySystem.dat': No such file or directory
ebylmz@ebylmz:~/cse/Operating-Systems/hw/hw03/src$ make run
gcc -Wextra -Wall makeFileSystem.c fs.c -o makeFileSystem
./makeFileSystem 4       mySystem.dat
superblock info
        fat position              : 20
        root directory position   : 8212
        data block position       : 90132
        num blocks                : 4096
        block size                : 4096
fat entries:
        entry no: 0 value: free
        entry no: 1 value: free
        entry no: 2 value: free
        entry no: 3 value: free
        entry no: 4 value: free
        entry no: 5 value: free
        entry no: 6 value: free
        entry no: 7 value: free
        entry no: 8 value: free
```

```
ebylmz@ebylmz: ~/cse/Operating-Systems/hw/hw03/src
        entry no: 4075 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4076 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4077 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4078 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4079 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4080 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4081 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4082 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4083 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4084 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4085 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4086 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4087 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4088 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4089 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4090 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4091 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4092 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4093 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4094 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
        entry no: 4095 name:  attr: 0 size: 0 last modf: 0 starting cluster 0
ebylmz@ebylmz:~/cse/Operating-Systems/hw/hw03/src$ ls -l mySystem.dat
-rw-rw-r-- 1 ebylmz ebylmz 16883732 Jun 18 22:44 mySystem.dat
ebylmz@ebylmz:~/cse/Operating-Systems/hw/hw03/src$
```

16MB disk file ‘mySystem.dat’ is created and its content printed on the console.