DESIGN DOC
Zac Plante
ASG4
CS 111

This is the design document for asg4 of CS111, the FAT file system. It will be divided by the different files in the directory, and each will be subdivided by its purpose and functions. Wishing a FAT file system, a file allocation table (FAT) is used to keep track of files that span multiple blocks. In the table, if the block contains the end of the file the value is -2, otherwise it is the index of the next block in the file. The 0 index block is the superblock with information about the file system, the next k blocks are the FAT, where k=number of blocks /(blocksize/4). This is because each block can fit blocksize/4 values, and you need on value for each block. The k+1 block is the root directory.

Superblock.h

Purpose

Superblock.h contains the Superblock struct for the FAT file system. This struct keeps track of the magic number (0xfa91283e as given by the assignment), the number of blocks in the FS, the number of blocks in the FAT, the size of each block (either 512B or 8KB), and the index of the root directory.

Directory.h

Purpose

Directory.h contains the Directory struct for the FAT file system. This struct keeps track of the file name, creation time, modification time, access time, the file length, the start index, and flags.

init_file.c

Purpose

init_file.c creates the disc image and initializes the superblock, the FAT, and the root directory and then places them at the appropriate disc image. It takes 3 arguments, the disc image file name which must be “disc.img”, the size of each block (512 or 8192 according to the assignment, but any is acceptable) and the number of blocks (>3).

writeblock()

writeblock() is a helper function that writes a buffer to a block at an index. It utilizes lseek() to go to the part of the file the block is located and puts it in the block.

readblock()

readblock() is a helper function that reads a buffer from a block at an index. It utilizes lseek() to go to the part of the file the block is located and puts it in the buff.

Fatsys.c

Purpose

Fatsys.c contains the functions that act allow the disc.img to be a FAT file system, rather then fuses passthrough system. It then has a struct to substitute the fuse functions for the FAT functions. It takes a single, the folder that you want to mount the file system on, and you must run it with sudo. It has three global variables, fd (the disc image file number), super (a pointer to the superblock), and block (an array of directory entries for directory blocks).

writeblock()

This function is identical to the one in init_file.c

readblock()

This function is identical to the one in init_file.c

getblock()

This function streamlines readblock(), using the fact that fd and super are global variables to only require the index and the buff as inputs.

putblock()

This function streamlines writeblock(), using the fact that fd and super are global variables to only require the index and the buff as inputs.

getname()

This function finds the last part of a path name so it can be used to put the name of a file or directory into the directory entry when making a new directory. So if the path is root/example/sample1, the function return sample1. It finds the index of the last / and putting everything after it into a return value.

findfreeblock()

findfreeblock() loops through the FAT blocks and all values in those blocks. When it finds a value of 0, it returns that index. If it cant find a free block, it return -1.

checkfat()

checkfat() takes and index and checks the FAT at that index and returns the value, either -1,-2,0, or a new index;

changefat()

changefat() is like checkout above, but it allows you to change the value.

findbypath()

This function has two parts. The first tokenizes the path into individual names for searching, the next loops through the directory looking for the current name and returns the index of it, as well as the block its in to block and the entry to dir. This function also has a flags option, where if you want the directory above the one requested in order to make a new file or directory, you can feed in a 1 instead of a 0;

FAT_open()

This function utilizes findbypath() to return the index of the file.

FAT_read()

This function reads a file into a buffer. It first finds the index of the file using find by path. After that, it starts using getblock() to get each block of the file and putting it into the buffer. It then gets the value at the same index in the FAT to continue reading using checkfat() for the whole size of the buffer. If at any point checkfat() returns -2, the function return -1 as the file is not big enough.

FAT_write()

This function writes a file from a buffer. It gets the index the same way the above function does with findbypath(). It then starts using putblock() to replace the index with the buffer for the whole size of the buffer. If they're are not enough blocks, the function allocates another block for the fie using findfreeblock() and updating the FAT.

FAT_init()

This is an initiation function. It defines the global variables super and block as being a pointer to the superblock and an array of directory entries respectively.

FAT_readdir()

This function buts all directory entries of a directory into a buffer. It uses findbypath() to get the directory. Then it loops through the block and adds all the entries. Then it checks if there is another block in the directory using checkfat(). As long as checkfat() isn't -2, there are more entries and the index changes to the value of checkfat().

FAT_getattr()

This function returns the attributes of a file and the file system designated by path. First, findbypath() gets the file designated by path. Then its attributes are put into the stat struct as well as attributes from super.

FAT_unlink()

This function removes a file from the data base. It uses findbypath() to get the directory entry and the block its in. Then it sets all values related to that entry in the in the FAT table to 0 to designate they are free. Finally it sets size of the directory entry to -1, which is my flag for showing this entry is free to be taken. After making these changes, it rewrites the block.

FAT_create()

This function makes a new file in the system. First it uses findbypath() with a flag of 1 to get to the parent directory. It then finds a free spot in the directory block to place a new directory entry. Then it gets the name with getname() and its index with findfreeindex(). After creating this new entry, it puts it in the block and writes the block back to the system.

FAT_mkdir()

This function behaves almost identically to FAT_create, but after making the entry, it goes to the new directory and creates two new entries, “.” which is a copy of the entry and “..” which is a copy of the parent directory.

FAT_rmdir()

This function works similarly to FAT_unlink(). It uses findbypath() to find the index of the directory block to remove. It sets this entry to size of -1 as i said in FAT_unlink(). Then it gets the block and recursively runs FAT_rmdir() if the entry is a directory and FAT_unlink() if its a file.

Makefile

Upon using make, two executables are created, init_file and Fatsys, use of these executables can be seen in their document as well as in the provided read me.