DV1628: Lab 3

Oscar Cederlund osce19@student.bth.se
Gustav Nilsson guni18@student.bth.se

January 2022

Contents

1 Private help functions

In order to make our life easier we've implemented a couple of extra functions that are used multiple times in the other functions.

1.1 accessrights_check

The function accessrights_check(int dir_entry_index, int accessrights) is used to check if we are allowed to read or write to a file or directory that is identified with the dir_entry_index. The action we want to do (Read or write) is identified with the acessrights int parameter. This function allows us to easy check different combinations of access rights when we want to interact with an item.

1.2 file_fit_check

The function file_fit_check(int num_blocks) checks if the fat can fit the amount of blocks we want to insert into it. This function is used everywhere when we want to write new blocks to the disk.

1.3 init_dir_content

This function is used to initialize the all the content that the root directory will hold. It went through a lot of different iterations as our way to implement a hierarchical system changed. For a long while it handled all directories an not just the root directory, but after many days of constant debugging we decided to rework it into it's current state. During this rework we also changed our current directory path variable dir_path from a vector of strings(directory names) to a vector of ints(directory dir_entry indexes).

At the moment however it works by finding all the dir_entry that's not a part of another directories content, thus marking them as orphans. We had earlier concluded that all the orphans must be a part of the root folder. The return value was something we struggles with a lot during the development as it at first was an int array similarly to the int array we used for our current directory content variable. This caused us a lot of issues with memory management which led us to replace it with a vector instead, which saved us a lot of trouble and headaches caused by the memory.

1.4 string_to_vector_converter

This is a function that transforms the input string into the int corresponding to the dir_entry index of the directory we're looking for as a part of task 4 and relative path. It does this by converting walking the hierarchical tree of dir_entries to our destination in mind.

2 Task 1

Here's how the functions from task 1 ended up in the finished code.

2.1 format

Format starts by setting our current path to the root, and writing empty strings to each block on the disk as well as marking the entire fat as FAT_FREE. After that we do a similar process to the dir_entries setting all the dir_entry structs in our dir_entries list to default values (Setting the name to an empty string, the int values to 0.) and all values in our curr_dir_content variable gets set to -1, our default "empty slot" value. Once all this is done we then proceed to set up our new fat and root block and writes them to the disk.

2.2 create

Our create function works by using by putting together the file content into one large string which we split up over multiple different blocks and we check if the needed amount of blocks will fit on the disk with our file_fit_check function. If it fits we proceed to write it into the fat (With each value inside the fat pointing to another fat index until we find the end of the file.) and into the blocks that found free.

Once the fat and the file content is done and handled we then proceed to to the files dir_entry setting the values and finding an index to save into our struct array dir_entries. When that is done we also save the index to our curr_dir_content so that we can find it with ease later.

2.3 cat

Cat works by checking allocating memory enough to hold the whole file content and then proceeds to read all the blocks on the disk by looking at the fat values. Once that is done prints out all of the content to the terminal and frees the allocated memory.

2.4 ls

Ls is a short one, it iterates through our curr_dir_content printing out all the file name, type, access rights and sizes of each entry.

3 Task 2

3.1 Cp

Our cp function works by finding the dir_entry index of the item we looking for, gather the amount of blocks that we need and check if they fits on the disk. We it's fine we proceed to a copy of original dir_entry, replacing some data such as the location for the first block with a new one that we get by looking at the fat which we also set up. Once is done we proceed with a process almost identical to the one in create were we write the file and all of its content to the disk, we put in the respective dir_content for the designation.

3.2 Mv

The mv function is all about moving the a dir_entry index from one directories dir_content to another directories dir_content and removing it from the original one. If we're moving the source to root directory we'll make it into an orphan by just removing it from it's current directory. When all the moving is done we save the directory contents of all the affected directories but the root(The root doesn't have an entry on the disk) to the disk.

3.3 Rm

Rm is similar to mv function in the sense that i removes the dir_entry index from the relevant dir_content array and then removes content from the disk, fat and root similar to the format, instead of saving the the new place like the mv function.

3.4 Append

The append function works by first gathering the information for the files we want to append to each other and concatenate the two strings. Then we create a new enlarge copy of the 2nd file which hold the new string, replacing the old one with the new file.

4 Task 3

4.1 mkdir

Mkdir is quite similar to the create function but instead a string stored in multiple blocks it stores a block containing an array called dir_content which contains all the dir_entries indexes for the different dir_entry variables that should be located in the directory.

4.2 cd

Cd works by switching out the content of the curr_dir_content for another directories dir_content array. For the root that doesn't have an array we instead use the init_dir_content function to find all the orphans. As mentioned earlier the new dir_content is stored in the block belonging to the directory.

4.3 pwd

Pwd is also quite a short and easy one, it reads all the file names from the dir_entry indexes stored in our dir_path variable, then it presents them neatly.

5 Task 4

To solve relative and absolute paths we created a function called string_to_vector_converter, that takes a path in string format(The one that you enter in the terminal) and gives you the directory index that you're looking for. This function as then been implemented in all the other functions of the program to handle the incoming paths.

6 Task 5

6.1 chmod

Chmod is also quite a simple function as much of the cases in the other functions are handled by the accessrights_check function. Chmod works by finding the dir_index of the file we're looking for and then sets the access_rights variable in the struct to the int that the combination of rights creates.