CS3026 - Assessment 2 Memory Management

The aim of this project is to implement a basic FAT filesystem with functionaly to manage files and directories.

File Structure

```
──cs3026_assessment2_AndrejSzalma
        ├── CGS_A5_A1
        │       ├── Makefile
        │       ├── filesys.c
        │       ├── filesys.h
        │       ├── shell.c
        │       ├── traceA5_A1.txt
        │       ├── virtualdiskA5_A1_a
        │       ├── virtualdiskA5_A1_b
        │       ├── virtualdiskA5_A1_c
        │       └── virtualdiskA5_A1_d
        ├── CGS_B3_B1
        │       ├── Makefile
        │       ├── filesys.c
        │       ├── filesys.h
        │       ├── shell.c
        │       ├── testfileB3_B1_copy.txt
        │       ├── traceB3_B1.txt
        │       ├── virtualdiskB3_B1
        │       ├── virtualdiskB3_B1_a
        │       └── virtualdiskB3_B1_b
        ├── CGS_C3_C1
        │       ├── Makefile
        │       ├── filesys.c
        │       ├── filesys.h
        │       ├── shell.c
        │       ├── testfileC3_C1_copy.txt
        │       ├── traceC3_C1.txt
        │       └── virtualdiskC3_C1
        ├── CGS_D3_D1
        │       ├── Makefile
        │       ├── filesys.c
        │       ├── filesys.h
        │       ├── shell.c
        │       └── virtualdiskD3_D1
        ├── CS3026_Assessment\ Instructions.pdf
        ├── README.md
        └── README.pdf
```

Every directory represents a group of source files based on the assesment instructions. Starting from the simplest functions implemented in CGS_D3_D1 the files are being expanded with new functions and their implementations up till CGS_A5_A1.

The source code in this project contains extensive comments which explain what is being done by the following lines of code. Hence, it will be not described in such detail in the report.

Usage & Specifications

This series of programs were developed to work with the `x86_64-linux-gnu` compiler. The `gcc version 7.5.0` was used to compile the files during the development.

Every folder contains a makefile, which can be run by the `make` command in an unix/linux terminal. This command will compile the source code and clean up after compilation. The resulting `shell` file is an executable which can be run by typing `./shell`.

CGS_D3_D1

The aim of this version of the program was to develop the `format()` function:

- `void format();` This function initializes our virtualdisk with 1024 blocks of size 1024 and the FAT table. The virtualdisk name is saved in the first block (lines 79:84), the FAT table is initialized and saved in the second and third blocks as it needs to have an entry for every block on the disk (lines87:95). A block is created at position 3 and it's first entry saved as "root" which indicates that this is going to be the root folder (lines 97:103). After that, the FAT table is updated because the root folder was added to block 3 and the rootDirIndex and currentDirIndex are set. Now the virtualdisk is ready for use.

Following is a `hexdump` of the virtualdiskD3_D1 showing the contents of the virtualdisk. It can be seen that the first block with the disk name is saved at the beggining of the virtualdisk. Following we can see two blocks containing the FAT table. At the beggining of this there is "00 00" symbolising the ENDOFCHAIN at FAT[0], the next record is "02 00" showing that FAT[1] is pointing to FAT[2]. The next "00 00" is again symbolising the ENDOFCHAIN at FAT[2] and the finall "00 00" is symbolising the ENDOFCHAIN at FAT[3] which is the location of the root folder. Now if we look at the third part (block) of the hexdump (starting at 0xc00), this is the block of the root folder, which can also be seen by the word "root".

```
00000000  43 53 33 30 32 36 20 4f  70 65 72 61 74 69 6e 67  |CS3026 Operating|
00000010  20 53 79 73 74 65 6d 73  20 41 73 73 65 73 6d 65  | Systems Assesme|
00000020  6e 74 20 32 20 2d 20 41  6e 64 72 65 6a 20 53 7a  |nt 2 - Andrej Sz|
00000030  61 6c 6d 61 00 00 00 00  00 00 00 00 00 00 00 00  |alma............|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400  00 00 02 00 00 00 00 00  ff ff ff ff ff ff ff ff  |................|
00000410  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00000c00  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000c10  00 00 00 00 00 00 00 00  00 00 00 00 00 00 72 6f  |..............ro|
00000c20  6f 74 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |ot..............|
00000c30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00100000
```

CGS_C3_C1

The aim of this version of the program was to develop file manipulation functionality:

- `MyFILE * myfopen( const char *, const char *);` takes a name and a manipulation mode string as an input. It looks in the directory entrylist for an entry with the filename. If it is found, it sets the file block number and buffer based on the record from the virtual disk. If there is no entry with the filename, a new entry is created in the entrylist, and a new block is allocated for the buffer. In both cases the file descriptor pointer is returned. There is an edge case of the directory being full, in that case a message is printed and NULL pointer returned.
- `void myfputc(int, MyFILE *);` takes an integer (which can also represent a character) and a file descriptor pointer. After that it sets the next free Byte to the integer from the input and increments the file descriptors pos property. If the end of a block is reached, a new block is allocated, pos reset and the buffer set to the new block.
- `void myfclose(MyFILE *);` takes a pointer to a file descriptor, inserts EOF marker at the end of the file buffer, saves all blocks that might have not been saved yet, and frees the memory of the file descriptor.
- `char myfgetc(MyFILE *);` takes a pointer to a file descriptor and returns the next Byte from the file buffer based on its pos property.

Following is a partial `hexdump` of the virtual disk as it has been cropped for the purposes of this paragraph. It can be seen that the virtual disk has extended by the testfile.txt file entry and its data. The FAT table has also been updated with new entries from FAT[4] -> FAT[9].

```
00000000  43 53 33 30 32 36 20 4f  70 65 72 61 74 69 6e 67  |CS3026 Operating|
00000010  20 53 79 73 74 65 6d 73  20 41 73 73 65 73 6d 65  | Systems Assesme|
00000020  6e 74 20 32 20 2d 20 41  6e 64 72 65 6a 20 53 7a  |nt 2 - Andrej Sz|
00000030  61 6c 6d 61 00 00 00 00  00 00 00 00 00 00 00 00  |alma............|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400  00 00 02 00 00 00 00 00  05 00 06 00 07 00 08 00  |................|
00000410  00 00 ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
00000420  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00000c00  01 00 00 00 00 00 00 00  00 00 00 00 00 01 00 00  |................|
00000c10  00 00 00 00 00 00 00 00  00 00 00 00 00 00 72 6f  |..............ro|
00000c20  6f 74 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |ot..............|
00000c30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000d20  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  |................|
00000d30  00 00 00 00 04 00 74 65  73 74 66 69 6c 65 2e 74  |......testfile.t|
00000d40  78 74 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |xt..............|
00000d50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001000  41 42 43 44 45 46 47 48  49 4a 4b 4c 4d 4e 4f 50  |ABCDEFGHIJKLMNOP|
00001010  51 52 53 54 55 56 57 58  59 5a 41 42 43 44 45 46  |QRSTUVWXYZABCDEF|
00001020  47 48 49 4a 4b 4c 4d 4e  4f 50 51 52 53 54 55 56  |GHIJKLMNOPQRSTUV|
00001030  57 58 59 5a 41 42 43 44  45 46 47 48 49 4a 4b 4c  |WXYZABCDEFGHIJKL|
```

CGS_B3_B1

The aim of this version of the program was to develop directory manipulation functionality:

- `void mymkdir (const char *);` takes a path string which can be absolute or relative. The function loops through the pathnames separated by "/" and looks if a directory with the name exists in the current directory. Subsequently it either sets the current directory to the found entry or creates a new directory. For this `findEntry(char *, dirblock_t *, char)` and `createDir(char *, dirblock_t *)` functions have been implemented.
    - `createDir(char *, dirblock_t *)` takes a directory name and a current directory pointer, finds an unused FAT entry and looks for a unused entry in the directories' entrylist. Then it allocates an empty block for the new directory and puts `.` and `..` as the first two entries in its entrylist. `.` is a pointer to itself and `..` is a

pointer to the parent directory. This functionality helps us to always know the properties of the current directory and its parent directory as well. Even though this functionality is required in the next grading bracket, it was natural for me to implement the directory structure this way. At the end of the function, the virtual disk address of the block allocated for the new directory is appended to the parent directories entrylist at the previously found unused entry position.

- `char ** mylistdir (char *);` takes a path string which can be absolute or relative and temporarily changes directory to it. For this the `void mychdir(char *)` function has been created which just keeps progressing through directories in the provided path, until it reaches the end and sets the currentDirIndex to the directory found, or reaches a non existent directory in which case it prints out an error message. After the correct directory has been set, the function loops through it's entry list and prints out all the entries with names longer than 0 while appending them to an array as well. A pointer to this char array is then returned and can be manipulated further if needed. It is worth noting that the `mychdir` function is required in the next grade bracket, however it was required to be able to create a file from a path not only filename, hence I found it only logical to create a function to change directories.

The above mention changes in the directory structure will also affect the look of the `hexdump`. A directory name will no longer show up as an entry in the directory entrylist, but there will be two entries corresponding to `.` and `..`.The `.` entry has a pointer to the directory name which is saved in the parent directories entrylist. This way we don't need to save the name twice, but we have access to it.

CGS_A5_A1

The aim of this version of the program was to develop the following:

- `void mychdir(char *);` as this function has been implemented in the previous block, it will not be discussed once more.
- `void myremove(const char *);` takes a string path as an input which can be absolute or relative and removes a file which name is assumed to be the last segment of the path. First, it progresses to the directory in which the file exists based on the path. It uses the same algorithm for splitting the path and filename as one used in `myfopen`. After the path is found, it changes to the directory and searches for the file. If the file is found, its blockchain is removed from the FAT table and the virtual disk and its entry in the parent directory entrylist removed. By doing this we ensure that there is no residual data left in the disk that could cause harm in the future.
- `void myrmdir(const char * path);` takes a string path as an input which can be absolute or relative and removes a directory if it is empty. At first it changes the current directory based on the provided path, checks if the directory is not the root directory (as that could not be removed) and also checks if the directory is empty. After that it finds the directory entry in the parent directories entrylist and removes the directories blockchain. When the blockchain is removed all the entries from the directories entrylist are removed as well and with that all traces of the directory are deleted.
- "implement '.' and '..' in the directory structure" - As this has been implemented in the previous block, it will not be discussed once more. However, a minor change was introduced in this version, where the dirblock does no longer have a pointer to the parent entries name, but the the whole parententry. This allows us to access more data while still taking up a little spae.
- "the function myfopen() can be called using an absolute or relative path in the filename" - This requirement seems rather unnecessary at this point as it basically had already implemented in the previous block. Nevertheless, it is fully functional and has been discussed previously.

A3_A1

Unfortunatelly, due to time constraints I have not managed to implement any part of this block.

Future possibilities

A very interesting and rewarding continuation of this project would be to create an interactive shell which would use unix-like commands to interpret our FAT filesystem.

Andrej Szalma - 51983729 - 05/12/2021