

CS3026 Assessment 2 Virtual Disk and FAT

Stefan Nikolov
November 19, 2019

1. CGS D3_D1

- Requirements: gcc compiler
- Instructions
 - Open terminal and navigate to the directory which contains the source files or **right-click** while in the folder and press **Open Terminal Here**.
 - Run the following command: **make**
 - Run the following command: **./shell**

- Explanation

First, we create the 0th block in our FAT table. Usually in real life situations it's reserved for high priority things (such as boot sector etc.) but in our case we just set its data to be "CS3026 Operating Systems Assessment" and then using the method writeblock() we save whatever is on the block to the virtual disk. Second, we prepare the FAT table. We initialize the 0th, 2nd and 3rd indices to be ENDOFCHAIN which act as ways to signal that the blocks are reserved or that a stream of bytes from a file is finished. We initialize the 1st index to the number 2 because that's where the FAT table resides. We proceed by setting each entry in the FAT table to UNUSED. Then using the method copyFAT() we transfer everything onto the virtual disk.

Third, we create a root directory block and an empty directory entry. We set the parameters of both accordingly and then we loop over each entry in said root directory and set it to the empty directory entry specified above.

After everything is done, we write the changes to root directory onto the virtual disk by using the writeblock() method once more.

- Output

The only real output to the console is a string stating the executing functionality.

The real output is seen in a so-called "hexdump".

This hexdump can be seen by running the following command in the directory in which the source code is: **hexdump -C virtualdiskD3_D1**

The expected output of running the command looks like the following

```

CS3026 Assessment 2 VirtualDisk : bash — Konsole
gcc -o shell filesystem.o shell.o
stefan@stefan-VirtualBox:~/C/Assignment2/CS3026 Assessment 2 VirtualDisk$ ./shell
(!) Execucting D3_D1 functionality
writedisk> virtualdisk[0] = CS3026 Operating Systems Assessment
stefan@stefan-VirtualBox:~/C/Assignment2/CS3026 Assessment 2 VirtualDisk$ hexdump -C virtualdiskD3_D
1
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 73 6d | Systems Assessm|
00000020 65 6e 74 00 00 00 00 00 00 00 00 00 00 00 00 00 |ent.....|
00000030 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 02 00 00 00 00 00 00 00 01 00 00 00 |.....|
00000c10 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 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
00000e30 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 |.....|
00000e40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
00100000
stefan@stefan-VirtualBox:~/C/Assignment2/CS3026 Assessment 2 VirtualDisk$

```

2. CGS C3_C1

- Requirements: gcc compiler
- Instructions
 - Open terminal and navigate to the directory which contains the source files or **right-click** while in the folder and press **Open Terminal Here**.
 - Run the following command: **make**
 - Run the following command: **./shell**
- Explanation
 - Myfopen()

When opening a file with myfopen() we have two possibilities. The first one is opening the file with the flag “r” (for reading) and the other one is with the flag “w” (for writing). In reality there are a couple more but those are the ones implemented in this program.

If the file is opened with the flag “r” a special utility method is called. This method goes through all directory entries in a buffer and it tries to find if there’s a file with the specified name. If there is such a file, it returns the beginning of the block in which it resides and if there isn’t it just returns FALSE.

If a file is returned by the utility method specified above, we return that file and exit out of the function.

If a file isn’t found, we inform the user that there has been an issue and exit out of the function.

If the file is opened with the flag “w” the same utility method is called to see whether or not such a file exists. If it does exist, this means that we will be overwriting it. First, we clear the FAT table of all entries which are associated with this file, then we iterate over disk blocks until we

find where the file resides. Once we've found the block, we proceed to empty it.

If, on the other hand, such a file doesn't exist we find the first unused block on the disk (by iterating over the FAT table) and store it to a variable. We also create a buffer block on which we store information taken from the virtual disk. We create a new entry and set its parameters to reflect that it's stored in the FAT table and set its entry point to be the first unused block. Once the FAT table has been dealt with, we move everything to the virtual disk accordingly and return the file for future use.

➤ **Myfputc()**

The first thing we do is check whether the file has been opened in the right mode or if it even exists. If either of the two criteria isn't met, we exit out.

Since our blocks have a size of 1024, we must always check whether or not we can continue writing on the current block or if we have to move to another one.

➤ **Myfgetc()**

This function is used to retrieve the next byte of information from an open file. If we have reached the end (indicated by ENDOFCHAIN) we return -1. If the end hasn't been reached, we first make sure we aren't trying to read information outside of the scope by setting the *pos* parameter of the file to 0 which resets the position of the pointer.

➤ **Myfclose()**

Used when we want to close a file. First, we check if such a file exists, if it doesn't, we just exit without doing anything. If a file does exist (and is opened) we write whatever is on the file to the virtual disk, so we don't lose any data and we proceed to close the file by freeing up the memory used by it.

- **Output**

The only real output to the console is a string stating the executing functionality.

The real output is seen in a so-called "hexdump".

This hexdump can be seen by running the following command in the directory in which the source code is: ***hexdump -C virtualdiskC3_C1***

The first thing we notice after running the command is that now there is a block in memory in which our file resides. We can see that by the name appearing in the hex. Further down we can see the data we've put into our file. For this example, I decided to loop over the English alphabet until 4KB of data has been written to the file. We can also see that no clutter is being shown when displaying the hexdump.

```

CS3026 Assessment 2 VirtualDisk : bash — Konsole
stefan@stefan-VirtualBox:~/C/Assignment2/CS3026 Assessment 2 VirtualDisk$ hexdump -C virtualdiskC3_C1
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 73 6d | Systems Assessm|
00000020 65 6e 74 00 00 00 00 00 00 00 00 00 00 00 00 00 |ent.....|
00000030 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 00 00 |.....|
00000410 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff |.....|
*
00000c00 01 00 00 00 02 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000c10 00 00 00 00 00 00 00 00 00 00 00 00 04 00 74 65 |.....te|
00000c20 73 74 66 69 6c 65 2e 74 78 74 00 00 00 00 00 00 |stfile.txt.....|
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 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
00000e30 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 |.....|
00000e40 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 5b 5c 5d 5e 5f 60 |QRSTUVWXYZABCDEF|
00001020 47 48 49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 |GHIJKLMNOPQRSTUW|
00001030 57 58 59 5a 5b 5c 5d 5e 5f 60 61 62 63 64 65 66 |VWXYZABCDEFGHIJKL|
00001040 4d 4e 4f 50 51 52 53 54 55 56 57 58 59 5a 5b 5c |MNOPQRSTUVWXYZAB|
00001050 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50 51 52 |CDEFGHIJKLMNOPQR|
00001060 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f 60 61 62 |STUVWXYZABCDEFGHI|
00001070 49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57 58 |IJKLMNOPQRSTUWVX|
00001080 59 5a 5b 5c 5d 5e 5f 60 61 62 63 64 65 66 67 68 |YZABCDEFGHIJKLMN|
00001090 4f 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e |OPQRSTUVWXYZABCD|
*
00001fb0 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50 51 52 |CDEFGHIJKLMNOPQR|
00001fc0 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f 60 61 62 |STUVWXYZABCDEFGHI|
00001fd0 49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57 58 |IJKLMNOPQRSTUWVX|
00001fe0 59 5a 5b 5c 5d 5e 5f 60 61 62 63 64 65 66 67 68 |YZABCDEFGHIJKLMN|
00001ff0 4f 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e |OPQRSTUVWXYZABCD|
00002000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
00100000
stefan@stefan-VirtualBox:~/C/Assignment2/CS3026 Assessment 2 VirtualDisk$

```

3. CGS B3_B1

- Requirements: gcc compiler
- Instructions
 - Open terminal and navigate to the directory which contains the source files or **right-click** while in the folder and press **Open Terminal Here**.
 - Run the following command: **make**
 - Run the following command: **./shell**
- Explanation
 - Mymkdir()

(!) NOTE (!) -> Creating files works only in the root directory. The functionality to create files in subsequent directories isn't implemented.

First, we check if we should set the current working directory to the root directory. Then we proceed to tokenize the path (this means we split the input string into smaller strings around a specified symbol

which in our case is “/” or DELIMITER as referred to in the source code). For each token we check if a directory with the same name already exists and if it does, we switch the current working directory to that one. So that we don’t end up overwriting directories and lose files.

If no such directory exists, we find the first unused block of memory in the disk and we transform that block into a block that describes a directory.

Then we create an empty directory block and an empty directory entry. We initialize them with the proper parameters and fill all entries in the directory block with the empty entry.

After that we write everything to the virtual disk and the FAT table.

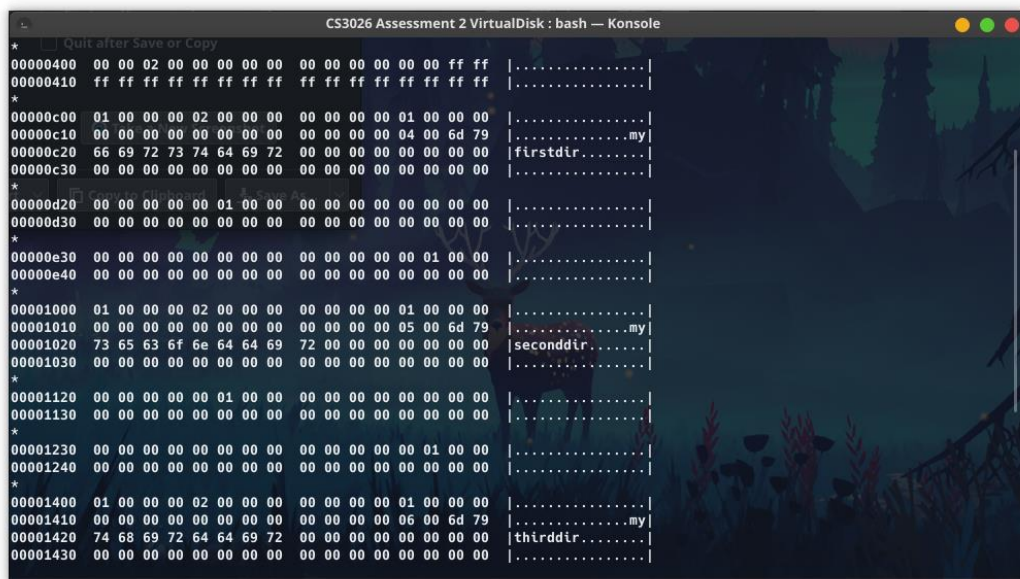
- Output

Again, the only output to the console when running the program is just a string indicating which functionality has been executed.

We must look at the hexdump

This hexdump can be seen by running the following command in the directory in which the source code is: **hexdump -C virtualdiskB3_B1**

The output of the command should show the 3 directories written in separate blocks and no clutter in between or anywhere as a matter of fact.



```
CS3026 Assessment 2 VirtualDisk : bash — Konsole
* | Quit after Save or Copy
00000400 00 00 02 00 00 00 00 00 00 00 00 00 ff ff |.....|
00000410 ff ff ff ff ff ff ff ff ff ff ff ff ff ff |.....|
*
00000c00 01 00 00 00 02 00 00 00 00 00 00 00 01 00 00 00 |.....|
00000c10 00 00 00 00 00 00 00 00 00 00 00 00 04 00 6d 79 |.....my|
00000c20 66 69 72 73 74 64 69 72 00 00 00 00 00 00 00 00 |firstdir.....|
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 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
00000e30 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 |.....|
00000e40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
00001000 01 00 00 00 02 00 00 00 00 00 00 00 01 00 00 00 |.....|
00001010 00 00 00 00 00 00 00 00 00 00 00 00 05 00 6d 79 |.....my|
00001020 73 65 63 6f 6e 64 64 69 72 00 00 00 00 00 00 00 |seconddir.....|
00001030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
00001120 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 |.....|
00001130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
00001230 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 |.....|
00001240 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
00001400 01 00 00 00 02 00 00 00 00 00 00 00 01 00 00 00 |.....|
00001410 00 00 00 00 00 00 00 00 00 00 00 00 06 00 6d 79 |.....my|
00001420 74 68 69 72 64 64 69 72 00 00 00 00 00 00 00 00 |thirddir.....|
00001430 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
```