CS3026 Assessment 2 FAT

Alexandar Dimitrov

51769662

Requirements – gcc compiler

Running Instructions:

To run each of the sections, follow the steps:

- open the terminal and navigate to the directory of the relevant section that contains the source files (cd CGS_D3_D1 for example)
- type “make” (to compile and link)
- type “./shell” (to run the relevant section)

To view the virtual disk in terminal, type the following:

- hexdump -C virtualdisk<relevant section (e.g. D3_D1)>

For part D:

```
hexdump -C virtualdiskD3_D1
```

For part C:

```
hexdump -C virtualdiskC3_C1
```

For part B:

```
hexdump -C virtualdiskB3_B1
```

For part A:

```
hexdump -C virtualdiskA5_A1
```

Sections:

GCD D3-D1

In this section I create the initial structure on the virtual disk by writing the FAT and the root directory into the virtual disk.

format() – the function creates the first 4 blocks of the virtual disk

- The root block occupying block 1
  - It is used to store the name of the disk

```
// implement format()
void format() {
    diskblock_t block;
    diskblock_t rootDir;
    int fatentry = 0;
    int pos = 0;
    int fatblocksneeded = (MAXBLOCKS / FATENTRYCOUNT);  // 1024/512 = 2

    /* prepare block 0: fill it with '\0',
     * use strcpy() to copy some text to it for testing purposes and
     * write block 0 to virtual disk
     */
    initializeBlock(&rootDir);      // initialize a rootDir block (set all its bytes to '\0')
    initializeBlock(&block);       // initialize a disk block (set all its bytes to '\0')
    strcpy(block.data, "CS3026 Operating Systems Assessment 2019");
    writeblock(&block, fatentry);   // writing to virtual disk
```

- The code above is the beginning the format function. It initialises 2 empty blocks ('block' and 'rootDir') by setting '/0' to every of their data position, copies the string (used to store the name of the disk – "CS3026 Operating Systems Assessment 2019") to the data section and write the block to index 0 of the virtual disk.

➢ The FAT blocks occupying blocks 2 and 3

- The code below initializes the FAT and writes it to the virtual disk via the copyFAT() helper function. CopyFAT() is explained on page 3.

```
/* prepare FAT table
 * write FAT blocks to virtual disk
 */
// -----------------------------------------------------------------------------------------------------
// Virtual Disk layout, i.e. known FAT layout:
FAT[0] = ENDOFCHAIN;         // block 0 is reserved and can contain any info about the whole file system on the disk
FAT[1] = 2;
FAT[2] = ENDOFCHAIN;        // blocks 1 and 2 are occupied by the FAT
FAT[3] = ENDOFCHAIN;        // block 3 is the root directory
//The rest of the virtual disk, blocks 4 - 1023, are either data or directory blocks:
for (pos=4; pos < MAXBLOCKS; pos++) FAT[pos] = UNUSED;
// -----------------------------------------------------------------------------------------------------
copyFAT(fatblocksneeded);   // Copy FAT to Virtual Disk via a helper function
```

The FAT occupies 2 blocks because each fat entry is 2 bytes of disk space, and with 1024 entries in the FAT it needs 2048 bytes, which are 2 blocks of disk space. Therefore, the chain of the FAT has a block chain of 2.

➢ The root directory block occupying block 4

- Directory block for the root directory is being created ('rootDir'). As the block is a directory, I set 'isdir' to 1(True). The block is then written to corresponding block number in the virtual disk. Finally, the 'rooDirIndex' as well as the 'currentDirIndex' is updated to corresponding position.

```
/* prepare root directory
* write root directory block to virtual disk
*/
// A directory block has an array of directory entries (which can be files and directories)
// and a "nextEntry" pointer that points to the next free list element for a directory entry
rootDir.dir.isdir = TRUE;
rootDir.dir.nextEntry = 0;      // set the first element in the entrylist
writeblock(&rootDir, 3);        // block 3 in the Virtual Disk is the root directory
rootDirIndex = 3;               // change rootDirIndex to index: fatblocksneeded+1
currentDirIndex = 3;            // change the currentDirIndex as well
```

➢ The rest of the blocks (5 – 1024) will be either data or directory blocks, they remain unused for now.

copyFAT() – the function copies the contents of the FAT into 2 blocks and then writes these 2 blocks to the virtual disk.

```
// copyFAT() helper function
void copyFAT(int fatblocksneeded) {
    // Copies the content of the FAT into one or more blocks (2 in my case),
    // then write these blocks to the virtual disk
    //Note: fatblocksneeded=2

    diskblock_t block;
    for (int blockIndex=0, i=0; blockIndex < fatblocksneeded; blockIndex++) {
        for (int j = 0; j < FATENTRYCOUNT; j++, i++) {
            block.fat[j] = FAT[i];          // copying the FAT into the 2 needed blocks
        }
        // write the blocks to the virtual disk
        writeblock(&block, blockIndex+1);  // block 0 is reserved, so we need blocks 1 and 2 who are occupied by the FAT
    }
}
```

As mentioned, the FAT structure will be divided into 2 blocks.
The outer loop iterates 2 times, while the inner – 512 times (FATENTRYCOUNT = BLOCKSIZE / sizeof(fatentry_t)). A FAT entry is 2 bytes (of *short* datatype) and the blocksize for the virtual disk is 1024. Therefore, we can store 512 FAT entries in one block of 1024 bytes. The FAT is managed as a local array and has 1024 bytes. However, it also has to be stored on the virtual disk. That is why the first half of FAT is written to the 2nd block and the second half to the 3rd block of the virtual disk.

Hexdump output for this part:

```
[root@localhost ~]# hexdump -C virtualdiskD3_D1
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 20 32 30 31 39  00 00 00 00 00 00 00 00  |ent 2019........|
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 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 00 00  |................|
*
00100000
[root@localhost ~]#
```

Explanation:
The first block occupies addresses 0x0 to 0x30. The hexadecimals corresponding to the characters (referring to the ASCII Table) are also visible – 43, 53, 33, etc. They represent the name of the disk.
The next 2 blocks are reserved for the FAT. FAT acts as a kind of block directory for the complete disk content. We can see the chain of the FAT – the first '00 00' at 0x400 is the entry of the FAT set to ENDOFCHAIN, the next '02 00' means that 2 blocks are reserved for the FAT showing the chain (block at index 1 has a value 2 and block at index 2 has a value 0, which is the ENDOFCHAIN). The rest remains UNUSED, which is -1 or 0xff. The last block in the virtual disk is the root directory. 01 at address 0x0c0 indicates this is a directory, isdir=1. The rest of block is initialized to '\0'.

GCD C3-C1

In this section a new file (‘textfile.txt’) is created in the FAT. This, subsequently, is reflected in the virtual disk. 4096 bytes are written to the file, resulting in 4 blocks being occupied in order to store the content. Then, those 4 blocks are read and written to a real hard disk file called ‘testfileC3_C1_copy.txt’.
The steps can be found in the ‘shell.c’ test file containing the calls to the functions powering the process. Namely those functions will be discussed below.

myfopen() – the main purpose of the function is to open a file and return it

```
// ------------------------------------------------------------------------------------------------------------------
// C part functions
MyFILE * myfopen(const char * filename, const char * mode) {
    // Opens a file on the virtual disk and manages a buffer for it of size BLOCKSIZE,
    // mode may be either “r” for readonly or “w” for read/write/append (default “w”).
    // IF the file does not exist and the mode is 'w' then a new file is being created,
    // otherwise ('r' mode) False is returned.

    if (strcmp(mode, "r") != 0 && strcmp(mode, "w") != 0){   // strcmp returns 0 if strings match
        printf("File mode not valid.\n");
        return FALSE;                                         // checking for correct mode
    }

    // Setting local variables
    int fatblocksneeded = (MAXBLOCKS / FATENTRYCOUNT);      // 1024/512 = 2
    bool existFile = false;                                 // boolean for checking if file already exists
    int pos;
    int freeListPos;
    diskblock_t block = virtualDisk[rootDirIndex];          // get directory entry block (block 4 at index 3 in the VD)

    MyFILE *file = malloc(sizeof(MyFILE));                  // allocating space for the file
```

Beginning of the myfopen() function

- If file is successfully opened, it is initialized with the built-in malloc() function
  - To be opened successfully file must be opened in a correct mode (only read ‘r’ and write ‘w’ modes are allowed)
- Then it is checked if the file exists in the root directory block of the virtual disk by comparing file names.

```
// Check for file existence in the directory block:
for (int i = 0; i < DIRENTRYCOUNT; i++) {
    // The Loop will iterate 3 times at most
    if (block.dir.entrylist[i].unused) continue;
    if (strcmp(block.dir.entrylist[i].name, filename) == 0) {
        pos = i;
        existFile = true;
        break;
    }
}
```

  - If the file does NOT exist and the fie mode is read (‘r’), False is returned because there is nothing to read from.
  - If the file does NOT exist and the file mode is write (‘w’), a new file is being created

```
// Write mode. File is being created.
for (int i = 0; i < DIRENTRYCOUNT; i++) {                  // Look for empty directory entry
    if (block.dir.entrylist[i].unused) {
        freeListPos = i;
        break;
    }
}

// Looks for a free fat entry; starts from index 4 as first 4 blocks are already reserved
for (pos = 4; pos < MAXBLOCKS; pos++) if (FAT[pos] == UNUSED) break;
FAT[pos] = ENDOFCHAIN;
copyFAT(fatblocksneeded);                                  // As stated each change in the FAT has to be copied to the Virtual Disk

strcpy(block.dir.entrylist[freeListPos].name, filename);   // Set the filename to the operating block
block.dir.entrylist[freeListPos].firstblock = pos;         // Set the position of first chain block in the directory list in accordance to the FAT table
block.dir.entrylist[freeListPos].unused = FALSE;           // Set file to 'used'
writeblock(&block, rootDirIndex);                          // Rewrite the root directory block to the virtual disk
file->blockno = pos;                                       // Set block number of file
file->buffer  = virtualDisk[pos];                          // Sets buffer to final block of file
```

For the file creation, an empty entry in the directory block has to be found to insert the file in. Also, a free FAT (FAT acts as a kind of block directory for the complete disk content) block has to be found for the same reason.
The free found FAT position is set to be the ENDOFCHAIN as the file is contained in no more than 1 block. Subsequently, **the change in the FAT is reflected in the virtual disk** – call copyFAT().
The filename and the first block in the chain are set. The found unused entry in the directory block is now in use. **The changes are reflected in the virtual disk** – call writeblock().
Finally, the file buffer and the file block number are set to the 1st block of the file chain.

- <u>If file exists:</u>
  Again, the file buffer and the file block number are set accordingly (to the 1st block of the file chain).

```
// If file already exists
if (existFile) {
    file->blockno = block.dir.entrylist[pos].firstblock;              // sets the file blockno being 1st block of the chain
    file->buffer  = virtualDisk[block.dir.entrylist[pos].firstblock]; // sets the buffer according to the virtual disk (the 1st block of the chain)
}
```

- If the file exists or the file does NOT exist, but is opened for a write operation, the file is returned opened.

```
// File is being returned, set the mode and the starting position
strcpy(file->mode, mode);                                  // Set the mode of the file
file->pos = 0;                                             // Set starting byte for file operations

return file;
```

Since the file mode and the cursor pointer (file->pos) are the same for both existing or non-existing files, they are set after all checks at the end of the function.

myfputc() – the function inserts a single character at a time to an opened file.

```
void myfputc(int b, MyFILE * stream) {
    // Used for WRITING.
    // Writes a byte to the file. Writes a block to the VD(Virtual Disk) when the block becomes full

    // Setting local variables
    int fatblocksneeded = (MAXBLOCKS / FATENTRYCOUNT);      // 1024/512 = 2
    int pos;

    diskblock_t * buffer = &stream->buffer;     // Get a pointer the the stream buffer (the start position of the file in fat)
    buffer->data[stream->pos] = b;              // Filling the data with content
    stream->pos++;                              // Incrementing the cursor pointer

    if (stream->pos >= BLOCKSIZE) {
        // When the buffer becomes full, the current buffer has to be written (copied) to the virtual disk and
        // a new block has to be allocated to the open file (an UNUSED block as indicated in the FAT)

        writeblock(buffer, stream->blockno);

        // Extending the Chain:
        for (pos = 4; pos < MAXBLOCKS; pos++) if (FAT[pos] == UNUSED) break;
        FAT[stream->blockno] = pos;             // continues the chain
        FAT[pos] = ENDOFCHAIN;                  // setting the end of the chain at this point of time
        copyFAT(fatblocksneeded);               // as stated each change in the FAT has to be copied to the Virtual Disk

        stream->blockno = pos;                  // moving the blockno
        stream->pos = 0;                        // resetting the cursor position
        for(int i = 0; i < BLOCKSIZE; i++) buffer->data[i] = '\0';
    }
}
```

The function uses a buffer that initially points to the start position of the file (the beginning of the chain) in the FAT to fill the data section with content. This happens slowly since just a single byte is inserted for each function call.
Each time a character is written to the content of the opened file, the cursor is incremented by one (stream->pos++). Then, there is a block of code that checks if the cursor will be too big for the current block on the next function call (greater than 1024 bytes).
If so, a couple of operations occur:

- the buffer content (all the 1024 written bytes) is copied to the corresponding block of the virtual disk.
- The chain is extended and copied to the VD
    - The current FAT entry is set to point to the next FAT entry
    - The next FAT entry is set to be the ENDOFTHECHAIN
    - New disk block is allocated to the open file – the next entry in the FAT with value UNUSED
    - The cursor position is reset (points again to 0)
    - The buffer data is cleared (all bytes set to '\0')

myfgetc() – the function reads and returns a single character at a time from an opened file.

```
int myfgetc(MyFILE * stream) {
    // Used for READING.
    // Returns the next byte of the open file, or -1 (EOF == -1)
    char result;

    // The first block of the file has to be loaded (copied) from the virtual disk when opened;
    // Each read pushes a position pointer to the end of the buffer, when it becomes BUFFERSIZE then the next block
    // from the virtual disk has to be loaded;
    // For this the chain in the FAT has to be followed to find the next block of the file
    stream->buffer = virtualDisk[stream->blockno];
    result = stream->buffer.data[stream->pos];              // Getting the character byte
    stream->pos++;                                          // Incrementing the cursor

    // If read operations go beyond the current buffer content, the next buffer according to the block chain in the FAT has to be loaded.
    if (stream->pos >= BLOCKSIZE) {
        if (FAT[stream->blockno] == ENDOFCHAIN) {           // End of chain is reached due to FAT (no more blocks to get)
            result = EOF;
        }
        stream->blockno = FAT[stream->blockno];             // Move blockNo to the next block in the FAT chain
        stream->pos = 0;                                    // Reset cursor
    }

    return result;                                          // Note: -1 could be returned due to end of the file content
}
```

The function uses a buffer that points to the relevant block of the file chain to read the data. The 'result' variable contains either the character that will be returned or the EOF if the end of the chain is reached. Note that the character could be -1 as the buffer runs out of content. Why? Because EOF(-1) is always inserted to the end of each file in the shell.c file. Before the result variable is returned at the end of the function, a block of code checks if the read operations go beyond the current buffer content (over 1024 bytes).

If so, a couple of operations occur:

- EOF check: If the end of the chain is not reached and there are more blocks to read information from, continue with the rest operations
- Cursor reset: the stream->pos (the cursor pointer) is set back to 0, so that we can start reading the next block in the chain.
- Adjust block: Move to the next block of the chain.

myfclose() – the function closes an opened file and writes any blocks not written to the virtual disk

```
void myfclose(MyFILE * stream){
    // Closes the file, writes out any blocks not written to disk
    writeblock(&stream->buffer, stream->blockno);
    free(stream);
}
```

Main point is saving a potential incomplete buffer because we wouldn't always have 1024-byte-filled blocks.

Functions flow in the shell file:

[WRITE] myfopen() -> myfputc() -> myfclose()

[READ] myfopen() -> myfgetc() -> myfclose()

Hexdump output for this part:

```
u15ad17@mb311h-l:~/Desktop/Assignment2 FAT/CGS_C3_C1$ 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 20 32 30 31 39  00 00 00 00 00 00 00 00  |ent 2019........|
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 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 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 00 01 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 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|
00001040  4d 4e 4f 50 51 52 53 54  55 56 57 58 59 5a 41 42  |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  41 42 43 44 45 46 47 48  |STUVWXYZABCDEFGH|
00001070  49 4a 4b 4c 4d 4e 4f 50  51 52 53 54 55 56 57 58  |IJKLMNOPQRSTUVWX|
00001080  59 5a 41 42 43 44 45 46  47 48 49 4a 4b 4c 4d 4e  |YZABCDEFGHIJKLMN|
00001090  4f 50 51 52 53 54 55 56  57 58 59 5a 41 42 43 44  |OPQRSTUVWXYZABCD|
000010a0  45 46 47 48 49 4a 4b 4c  4d 4e 4f 50 51 52 53 54  |EFGHIJKLMNOPQRST|
000010b0  55 56 57 58 59 5a 41 42  43 44 45 46 47 48 49 4a  |UVWXYZABCDEFGHIJ|
000010c0  4b 4c 4d 4e 4f 50 51 52  53 54 55 56 57 58 59 5a  |KLMNOPQRSTUVWXYZ|
000010d0  41 42 43 44 45 46 47 48  49 4a 4b 4c 4d 4e 4f 50  |ABCDEFGHIJKLMNOP|
000010e0  51 52 53 54 55 56 57 58  59 5a 41 42 43 44 45 46  |QRSTUVWXYZABCDEF|
000010f0  47 48 49 4a 4b 4c 4d 4e  4f 50 51 52 53 54 55 56  |GHIJKLMNOPQRSTUV|
00001100  57 58 59 5a 41 42 43 44  45 46 47 48 49 4a 4b 4c  |WXYZABCDEFGHIJKL|
00001110  4d 4e 4f 50 51 52 53 54  55 56 57 58 59 5a 41 42  |MNOPQRSTUVWXYZAB|
00001120  43 44 45 46 47 48 49 4a  4b 4c 4d 4e 4f 50 51 52  |CDEFGHIJKLMNOPQR|
00001130  53 54 55 56 57 58 59 5a  41 42 43 44 45 46 47 48  |STUVWXYZABCDEFGH|
00001140  49 4a 4b 4c 4d 4e 4f 50  51 52 53 54 55 56 57 58  |IJKLMNOPQRSTUVWX|
```

……

to the last 8th block of the file chain

……

```
00001f70  41 42 43 44 45 46 47 48  49 4a 4b 4c 4d 4e 4f 50  |ABCDEFGHIJKLMNOP|
00001f80  51 52 53 54 55 56 57 58  59 5a 41 42 43 44 45 46  |QRSTUVWXYZABCDEF|
00001f90  47 48 49 4a 4b 4c 4d 4e  4f 50 51 52 53 54 55 56  |GHIJKLMNOPQRSTUV|
00001fa0  57 58 59 5a 41 42 43 44  45 46 47 48 49 4a 4b 4c  |WXYZABCDEFGHIJKL|
00001fb0  4d 4e 4f 50 51 52 53 54  55 56 57 58 59 5a 41 42  |MNOPQRSTUVWXYZAB|
00001fc0  43 44 45 46 47 48 49 4a  4b 4c 4d 4e 4f 50 51 52  |CDEFGHIJKLMNOPQR|
00001fd0  53 54 55 56 57 58 59 5a  41 42 43 44 45 46 47 48  |STUVWXYZABCDEFGH|
00001fe0  49 4a 4b 4c 4d 4e 4f 50  51 52 53 54 55 56 57 58  |IJKLMNOPQRSTUVWX|
00001ff0  59 5a 41 42 43 44 45 46  47 48 49 4a 4b 4c 4d 4e  |YZABCDEFGHIJKLMN|
00002000  ff 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00100000
u15ad17@mb311h-l:~/Desktop/Assignment2 FAT/CGS_C3_C1$
```

Explanation:

The hexdump is based on the D part hexdump (you can see its explanation on page 3). However, there are some differences that show the changes made in this part. Now, there is a visible chain the FAT. The 4096 bytes written to the created file, form a chain (visible at 0x400) – the content of the file 'testfile.txt' starts at block5 (at index 4), so FAT entry 04 contains 05 00 (don't forget each FAT entry is 2 bytes), entry 05 00 contains 06 00, etc. until 08 00. The hexadecimal values of the letters of the file content, stored in the memory addresses of the blocks are also visible. Note the 'ff' at 0x2000 – it shows the inserted EOF(-1) to the end of the file.

GCD B3-B1

In this section, I create a new path with the mymkdir() function and then list the contents of a subpath, printing it out via the mylistdir() function – this is saved to virtualdiskB3_B1_a. Then, I create a file (again with the mymkdir() function) in the virtual disk and list the contents of the same subpath, printing it out via the mylistdir() function – this is saved to virtualdiskB3_B1_b.
The steps can be found in the 'shell.c' test file containing the calls to the functions powering the process. Namely those functions will be discussed below.

mymkdir() – the function creates a directory from path
If the directories in the path hierarchy do not exist, they are created accordingly.

```
void mymkdir(const char * path){
    // Creates a new directory, using path, e.g. mymkdir ("/first/second/third") creates directory "third" in parent dir "second",
    // which is a subdir of directory "first", and "first" is a sub directory of the root directory.

    // Setting local variables
    int fatblocksneeded = (MAXBLOCKS / FATENTRYCOUNT);      // 1024/512 = 2
    int pos;
    int freeEntryIndex;
    char *copy = strdup(path);                              // copy of the path string
    char *remainingPath = NULL;

    diskblock_t block = virtualDisk[currentDirIndex];       // get current directory block
    if (path[0] == '/') {                                   // If absolute path:
        block = virtualDisk[rootDirIndex];                  // get root block
    }


    char *dirName = strtok_r(copy, "/", &remainingPath);    // strtok_r maintains an internal variable for the parsed string.
    // Split the given path at the "/" character.
    while(dirName){                                         // loop over dirs in path with strtok_r while there are still remaining tokens
```

The Beginning of the mymkdir() function

Initially, the function starts at the relevant current directory thanks to the global variable 'currentDirIndex', however if path starts with '/' – the start is at the root directory. In both cases the block at this index from the virtual disk is addressed and used. Then, an iteration over the path begins, tokenizing the path string using the strtok_r function, part of the c standard library. The built-in function splits the path at the '/' character in order to extract the names of the directories and loop over each of them.
After that, an inner loop iterates through the directory entries in the relevant block comparing names to see if their name matches the current directory name extracted from the strtok_r function. The process can be seen below:

```
bool existDir = false;                          // boolean for checking if dir already exists
//Check if dir already exists:
for(int i = 0; i < DIRENTRYCOUNT; i++) {
    if (strcmp(block.dir.entrylist[i].name, dirName) == 0) {
        currentDirIndex = block.dir.entrylist[i].firstblock;    // increment currDirIndex by 1 (move to the next)
        block = virtualDisk[currentDirIndex];       // pick the next block
        existDir = true;                            // dir has been found
        break;
    }
}
```

- If there is a match, go down one directory level because there is no need to create a directory which has already been created.
- If there is no match, the outer loop (over the path directory names) continues with the creation of the path directory.

```
if(!existDir) {
    freeEntryIndex = findEmptyDirEntryPos(&block);              // get the next free entry in the entrylist
    // Looks for a free fat entry; starts from index 4 as first 4 blocks are already reserved
    for (pos = 4; pos < MAXBLOCKS; pos++) if (FAT[pos] == UNUSED) break;
    FAT[pos] = ENDOFCHAIN;                                      // FAT position now occupied

    block.dir.isdir = true;                                     // mark block as a directory
    strcpy(block.dir.entrylist[freeEntryIndex].name, dirName);  // dir name now set to the operating block
    block.dir.entrylist[freeEntryIndex].firstblock = pos;       // set position of the first block in the dir list in accordance to the FAT table
    block.dir.entrylist[freeEntryIndex].unused = FALSE;         // dir entry now in use

    printf("Dir '%s' has been created.\n", dirName);

    writeblock(&block, currentDirIndex);                        // saves to the VD
    currentDirIndex = pos;                                      // increment the currentDirIndex
    block = virtualDisk[currentDirIndex];                       // get the next VD free block
    // Prepare the new block dir entries
    for (int i = 0; i < DIRENTRYCOUNT; i++) block.dir.entrylist[i].unused = TRUE;
    writeblock(&block, currentDirIndex);                        // saves to the VD
}
dirName = strtok_r(NULL, "/", &remainingPath);          // NULL indicates that strtok_r should return the next token(dir name)
}
```

In order to properly create the directory, the FAT and the VD have to be used. I firstly find an unused directory entry in the entrylist of the VD block that I operate on (via the 'findEmptyDirEntryPos' helper function). Then, I also find the next unused FAT entry. Next, a couple of operations occur:

- 'isdir' property is set to true so that it can be visible in the hexdump
- Once I find the unused directory entry space, I:
    - set its position to used
    - update the entry list entry for the new sub directory to point to the newly found space (in accordance to the FAT) for the sub directory.
    - set the name of the directory I wish to create into that space of the current block
- The directory block is written to the virtual disk
    - Note that the FAT is also written to the VD at the very end of the function, although is not visible in the block of code above.

Finally, the currentDirIndex is set to the new-found free FAT entry position (go down one level). This is the new current working directory, which entrylist entries have to be set to unused and saved to the VD.

The process will repeat until all sub directories of the path are processed. When the token = NULL.

mylistdir() – this function returns the content of a directory given a path that is accepted as a parameter

```
char ** mylistdir(const char * path){
    //  The Function lists the content of a directory and returns a list of strings.

    // Setting local variables
    char *copy = strdup(path);                          // copy of the path string
    char *remainingPath = NULL;
    diskblock_t block = virtualDisk[rootDirIndex];      // get directory root block (block 4 at index 3 in the VD)
    bool existDir;                                      // boolean for checking if dir already exists

    // Declare a pointer to a pointer to char and then allocate an array of pointers to character with DIRENTRYCOUNT elements
    char ** myStringListPointer;
    // This allocates an array of pointers to character with DIRENTRYCOUNT elements.
    // Allocating memory for pointers, as myStringListPointer is a list of pointers:
    myStringListPointer = malloc(sizeof(char *) * DIRENTRYCOUNT) ;


    char *dirName = strtok_r(copy, "/", &remainingPath);    // strtok_r maintains an internal variable for the parsed string.
```

The Beginning of the mylistdir() function

The function goes through a similar process as in mymkdir() function but as opposed to the mymkdir() function, mylistdir() does not create new directories.
Again, there is an outer loop that iterates through the names of the directories from the path using the strtok_r function and an inner loop that iterates through the directory entries in the relevant block comparing names to see if their name matches the current directory name extracted from the strtok_r function. If there is a match, I go down one directory level. If there is no match, “Path not found” is returned. Once all directories are processed, I loop through entries in current directory and add them to the variable that is being returned.

```
if(existDir) {
    // Being in the current directory, I extract the content of the directory
    // Allocate memory for each of its files/dir names and then copy the names
    for (int i = 0; i < DIRENTRYCOUNT; i++) {
        myStringListPointer[i] = malloc(sizeof(char)*(strlen(block.dir.entrylist[i].name)+1));
        strcpy(myStringListPointer[i], block.dir.entrylist[i].name);
    }
}
return myStringListPointer;
```

Hexdump output for this part:

```
u15ad17@mb311h-l:~/Desktop/Assignment2 FAT/CGS_B3_B1$ hexdump -C virtualdiskB3_B1_a
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 20 32 30 31 39  00 00 00 00 00 00 00 00  |ent 2019........|
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  00 00 00 00 00 00 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 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 00 01 00 00  |................|
00000e40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001000  01 00 00 00 00 00 00 00  00 00 00 00 00 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 00 01 00 00  |................|
00001240  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001400  01 00 00 00 00 00 00 00  00 00 00 00 00 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  |................|
*
```

virtualdiskB3_B1_a

```
u15ad17@mb311h-l:~/Desktop/Assignment2 FAT/CGS_B3_B1$ hexdump -C virtualdiskB3_B1_b
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 20 32 30 31 39  00 00 00 00 00 00 00 00  |ent 2019........|
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  00 00 00 00 00 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 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 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 00 01 00 00  |................|
00000e40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001000  01 00 00 00 00 00 00 00  00 00 00 00 00 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 00 01 00 00  |................|
00001240  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001400  01 00 00 00 00 00 00 00  00 00 00 00 00 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  |................|
*
00001530  00 00 00 00 07 00 74 65  73 74 66 69 6c 65 2e 74  |......testfile.t|
00001540  78 74 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |xt..............|
00001550  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

virtualdiskB3_B1_b

GCD A5-A1

For this section, I modified myfopen() so that files can be created accordingly and also implemented all of the required functions. The relevant virtualdiskA5_A1_a..d files have also been created. The 'shell.c' test file guides the user through a series of operations and contains the calls to the functions. Namely those functions will be discussed below.

Modification in the myfopen():

```
// ------------------------------------ A part modifications ---------------------------------------------------
// Check the number of tokens (the number of the dir names + filenames
char *path = strdup(filename);                          // copy of the path filename string
int counter = 0;
for(int i=0; i<strlen(path); i++) if(path[i] == '/') counter++;

// Check if the number of tokens in the path are more than 1
//   1) If more then 1 (e.g /firstdir/seconddir/file.txt)
//         • Call the mkdir to create the missing dirs (if any).
//         • Extract the name of the file from the path, set it to the filename variable
//           and continue the myfopen function as normal
//   2) If just 1 (the name of the file to be created):
//         • Continue the myfopen() function as normal to create the file
if(counter >= 1) {
    printf(".......");
    mymkdir(path);
    filename = strrchr(path, '/') + 1;
}

// diskblock_t block = readblock(rootDirIndex);         // previously
// Changed to:
diskblock_t block = virtualDisk[currentDirIndex];       // get current directory block
// ------------------------------------ A part modifications ---------------------------------------------------
```

myrmdir() - this function removes an existing directory, using path

Note1: It **can only** remove directories (files are removed by calling myremove)
Note2: It **cannot** remove a directory which has subfolders/subfiles.
The function goes through a similar iterative process as in mymkdir() function but as opposed to the mymkdir() function, myrmdir() does not create new directories but deletes them.
Here is how the directory do be deleted is found in the iterative process:

```
bool existDir = false;                          // boolean for checking if dir already exists
// Check if dir already exists:
for(int i = 0; i < DIRENTRYCOUNT; i++) {
    if (strcmp(block.dir.entrylist[i].name, dirName) == 0) {
        location = i;
        parentIndex = currentDirIndex;              // track the parentIndex
        currentDirIndex = block.dir.entrylist[i].firstblock;    // update currentDirIndex
        block = virtualDisk[currentDirIndex];       // pick the next block (go down 1 dir level)
        existDir = true;                            // dir has been found
        copyDirName = dirName;
        break;
    }
}

if(!existDir) {
    printf("Path not found.\n");
    return;
}
```

Once found, the directory is being deleted (under some constraints), shown below:

```
// Check if you are about to delete a directory or a file by checking the isdir property.
//   1) If dir (isdir = true)        -> Delete dir
//   2) If file (isdir = false)      -> Show error message
diskblock_t previous_block = virtualDisk[parentIndex];                  // pick the parent block
if(previous_block.dir.entrylist[location].isdir) {
    empty = true;
    for (int i = 0; i < DIRENTRYCOUNT; i++) {
        // Check if the directory entries are empty
        //   1) If not empty   -> Cannot remove a directory with files/dirs in it
        //   2) If empty       -> Delete
        if (!block.dir.entrylist[i].unused) {
            printf("Cannot remove a directory which has subfolders/subfiles.\n");
            empty = false;       // prevent from the deletion
        }
    }

    if (empty) {
        FAT[currentDirIndex] = UNUSED;              // Free the FAT
        copyFAT(fatblocksneeded);                   // Save the changes

        // Remove the directory
        direntry_t emptyDirEntry;
        for (int j = 0; j < MAXNAME; j++) emptyDirEntry.name[j] = '\0';
        virtualDisk[parentIndex].dir.entrylist[location] = emptyDirEntry;    // delete the dir name
        virtualDisk[parentIndex].dir.entrylist[location].unused = TRUE;      // set to unused
        printf("Dir %s sucessfully removed!\n", copyDirName);
    }
}
else{
    printf("Cannot remove a file with myrmdir(). Try myremove() instead.\n");
};
```

myremove() – this function removes an existing file

Note: It **can only** remove files (directories are removed by calling myrmdir)

The function goes through a similar iterative process as in previous functions.

The function finds the file do be deleted and remembers the index to be deleted:

```
for(int i = 0; i < DIRENTRYCOUNT; i++) {
    if (strcmp(block.dir.entrylist[i].name, dirName) == 0) {
        location = i;
        deletedIndex = currentDirIndex;              // track the index to be deleted
        currentDirIndex = block.dir.entrylist[i].firstblock;    // update currentDirIndex
        block = virtualDisk[currentDirIndex];        // pick the next block (go down 1 dir level)
        existFile = true;                            // file has been found
        copyDirName = dirName;
        break;
    }
```

Once found, the file is being deleted (under some constraints), shown below:

```
  // Check if you are about to delete a directory or a file by checking the isdir property.
  //   1) If dir (isdir = true)        -> Show error message
  //   2) If file (isdir = false)      -> Delete file
  // Remove the file
  diskblock_t previous_block = virtualDisk[deletedIndex];              // pick the parent block
  if (!previous_block.dir.entrylist[location].isdir) {
      FAT[currentDirIndex] = UNUSED;              // Free the FAT
      copyFAT(fatblocksneeded);                   // Save the changes
      direntry_t emptyDirEntry;
      for (int j = 0; j < MAXNAME; j++) emptyDirEntry.name[j] = '\0';
      virtualDisk[deletedIndex].dir.entrylist[location] = emptyDirEntry;    // delete the file name
      virtualDisk[deletedIndex].dir.entrylist[location].unused = TRUE;      // set to unused
      printf("File %s sucessfully removed!\n", copyDirName);
  }
  else{
      printf("Cannot remove directories with myremove(). Try myrmdir() instead.\n");
  }
```

mychdir() - this function will change into an existing directory, using path

The function goes through a similar process (actually the very same) as in mymkdir() function but as opposed to the mymkdir() function, which create new directories, mychdir() sets the current directory once all directories have been processed. In fact, the iterating process is pretty much the same as in the rest of the functions of part B and A and could have been moved to a new helper function that is being called.

Hexdump output for this part:

```
u15ad17@mb311h-l:~/Desktop/Assignment2 FAT/CGS_A5_A1$ hexdump -C virtualdiskA5_A1_a
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 20 32 30 31 39  00 00 00 00 00 00 00 00  |ent 2019........|
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  00 00 00 00 00 00 00 00  |................|
00000410  00 00 00 00 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 01 00 00 00  |................|
00000c10  00 00 00 00 00 00 00 00  00 00 00 00 04 00 66 69  |..............fi|
00000c20  72 73 74 64 69 72 00 00  00 00 00 00 00 00 00 00  |rstdir..........|
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 00 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 73 65  |..............se|
00001020  63 6f 6e 64 64 69 72 00  00 00 00 00 00 00 00 00  |conddir.........|
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 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00001410  00 00 00 00 00 00 00 00  00 00 00 00 05 00 74 65  |..............te|
00001420  73 74 66 69 6c 65 31 2e  74 78 74 00 00 00 00 00  |stfile1.txt.....|
00001430  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001530  00 00 00 00 05 00 74 65  73 74 66 69 6c 65 32 2e  |......testfile2.|
00001540  74 78 74 00 00 00 00 00  00 00 00 00 00 00 00 00  |txt.............|
00001550  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001630  00 00 00 00 00 00 00 00  00 00 00 00 01 00 00 00  |................|
00001640  00 00 00 00 00 00 00 00  00 00 00 00 08 00 74 68  |..............th|
00001650  69 72 64 64 69 72 00 00  00 00 00 00 00 00 00 00  |irddir..........|
00001660  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001800  54 68 69 73 20 69 73 20  74 68 65 20 66 69 72 73  |This is the firs|
00001810  74 20 66 69 6c 65 20 63  6f 6e 74 65 6e 74 ff 00  |t file content..|
00001820  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001c00  54 68 69 73 20 69 73 20  74 68 65 20 73 65 63 6f  |This is the seco|
00001c10  6e 64 20 66 69 6c 65 20  63 6f 6e 74 65 6e 74 ff  |nd file content.|
00001c20  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002010  00 00 00 00 00 00 00 00  00 00 00 00 08 00 74 65  |..............te|
00002020  73 74 66 69 6c 65 33 2e  74 78 74 00 00 00 00 00  |stfile3.txt.....|
00002030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002120  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  |................|
00002130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002230  00 00 00 00 00 00 00 00  00 00 00 00 01 00 00 00  |................|
00002240  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002400  54 68 69 73 20 69 73 20  74 68 65 20 74 68 69 72  |This is the thir|
00002410  64 20 66 69 6c 65 20 63  6f 6e 74 65 6e 74 ff 00  |d file content..|
00002420  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00100000
u15ad17@mb311h-l:~/Desktop/Assignment2 FAT/CGS_A5_A1$
```

***virtualdiskA5_A1_a** after the process of creation*

```
u15ad17@mb311h-l:~/Desktop/Assignment2 FAT/CGS_A5_A1$ hexdump -C virtualdiskA5_A1_d
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 20 32 30 31 39  00 00 00 00 00 00 00 00  |ent 2019........|
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 00 00 00 00  |................|
00000410  ff ff 00 00 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  ff ff ff ff ff 01 ff ff  |................|
00000c10  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff 00 00  |................|
00000c20  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000d10  00 00 00 00 00 00 00 00  00 00 00 00 00 00 ff ff  |................|
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 00 01 00 00  |................|
00000e40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001000  01 00 00 00 00 00 00 00  ff ff ff ff ff 01 ff ff  |................|
00001010  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff 00 00  |................|
00001020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 ff ff  |................|
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 00 00 00 00  ff ff ff ff ff 01 ff ff  |................|
00001410  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff 00 00  |................|
00001420  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001510  00 00 00 00 00 00 00 00  00 00 00 00 00 00 ff ff  |................|
00001520  ff ff ff ff ff 01 ff ff  ff ff ff ff ff ff ff ff  |................|
00001530  ff ff ff ff ff ff 00 00  00 00 00 00 00 00 00 00  |................|
00001540  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001630  00 00 00 00 00 00 ff ff  ff ff ff ff 01 ff ff ff  |................|
00001640  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff 00 00  |................|
00001650  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001740  00 00 00 00 00 00 00 00  00 00 00 00 00 00 ff ff  |................|
00001750  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001800  54 68 69 73 20 69 73 20  74 68 65 20 66 69 72 73  |This is the firs|
00001810  74 20 66 69 6c 65 20 63  6f 6e 74 65 6e 74 ff 00  |t file content..|
00001820  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001c00  54 68 69 73 20 69 73 20  74 68 65 20 73 65 63 6f  |This is the seco|
00001c10  6e 64 20 66 69 6c 65 20  63 6f 6e 74 65 6e 74 ff  |nd file content.|
00001c20  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002000  00 00 00 00 00 00 00 00  ff ff ff ff ff 01 ff ff  |................|
00002010  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff 00 00  |................|
00002020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 ff ff  |................|
00002120  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  |................|
00002130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002230  00 00 00 00 00 00 00 00  00 00 00 00 01 00 00 00  |................|
00002240  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002400  54 68 69 73 20 69 73 20  74 68 65 20 74 68 69 72  |This is the thir|
00002410  64 20 66 69 6c 65 20 63  6f 6e 74 65 6e 74 ff 00  |d file content..|
00002420  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00100000
```

***virtualdiskA5_A1_d** after the process of deletion*

Output of the ./shell file:

```
Call function mymkdir("/firstdir/seconddir"):
'firstdir' has been created.
'seconddir' has been created.

Call function myfopen("firstdir/seconddir/testfile1.txt", "w"):
.......
Call function mymkdir("firstdir/seconddir/testfile1.txt"):
'testfile1.txt' has been created.
Write 30 bytes to the file and close the file.

Call function mylistdir("/firstdir/seconddir"):
---> testfile1.txt

Call function mychdir("/firstdir/seconddir"):
Current Dir Index: 5

Call function myfopen("testfile2.txt", "w"):
'testfile2.txt' has been created.
Write 31 bytes to the file and close the file.
```

```
Call function mymkdir("/firstdir/seconddir/thirddir"):
'thirddir' has been created.

Call function myfopen("thirddir/testfile3.txt", "w"):
.......
Call function mymkdir("thirddir/testfile3.txt"):
'testfile3.txt' has been created.
Write 30 bytes to the file and close the file.

Call function mylistdir("/firstdir/seconddir"):
---> testfile1.txt
---> testfile2.txt
---> thirddir

Call function mylistdir("/firstdir/seconddir/thirddir"):
---> testfile3.txt
writedisk> virtualdisk[0] = CS3026 Operating Systems Assessment 2019

================================================================================

Call function myrmdir("/firstdir/seconddir/testfile1.txt"):
Cannot remove a file with myrmdir(). Try myremove() instead.

Myremove():
Call function myremove("/firstdir/seconddir/testfile1.txt"):
File testfile1.txt sucessfully removed!

Call function mychdir("/firstdir/seconddir"):
Current Dir Index: 5

Call function myremove("testfile2.txt"):
File testfile2.txt sucessfully removed!
writedisk> virtualdisk[0] = CS3026 Operating Systems Assessment 2019

================================================================================

Call function mychdir("thirddir"):
Current Dir Index: 8

Call function myrmdir("/firstdir/seconddir/thirddir"):
Cannot remove a directory which has subfolders/subfiles.

Call function mylistdir("/firstdir/seconddir/thirddir"):
---> testfile3.txt

Call function myremove("testfile3.txt"):
File testfile3.txt sucessfully removed!
writedisk> virtualdisk[0] = CS3026 Operating Systems Assessment 2019

================================================================================

Call function mychdir("/firstdir/seconddir"):
Current Dir Index: 5

Call function mylistdir("/firstdir/seconddir"):
---> thirddir

Call function myrmdir("thirddir"):
Dir thirddir sucessfully removed!
```

```
Call function mychdir("/firstdir"):
Current Dir Index: 4

Call function myremove("seconddir"):
Cannot remove directories with myremove(). Try myrmdir() instead.

Call function myrmdir("/firstdir/seconddir"):
Dir seconddir sucessfully removed!

Call function mychdir("/"):
Current Dir Index: 3

Call function myrmdir("firstdir"):
Dir firstdir sucessfully removed!
writedisk> virtualdisk[0] = CS3026 Operating Systems Assessment 2019

Process finished with exit code 0
```

Conclusions:

Given more time, I would think of more edge-cases throughout the whole program, so that they can be properly addressed.