EE442 HOMEWORK 4

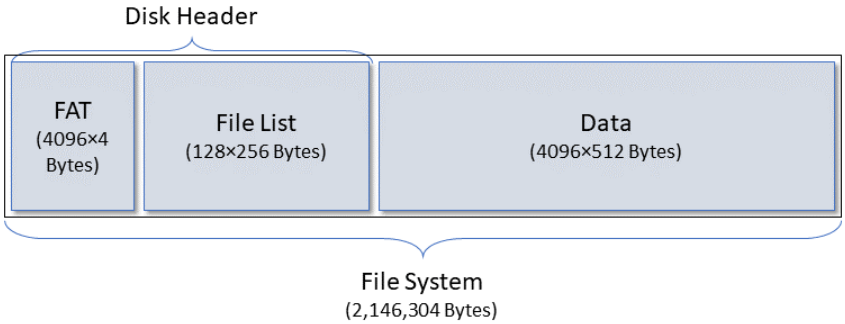
File System

Due: May 29, 2017, 23:55

Okan Çalış calis.okan@metu.edu.tr

Description:

In this homework, you are expected to implement a simple file system that uses a file allocation table (FAT). The resultant file system will be employed in a physical drive or a disk image.



The file system is expected to consist of three parts:

1. File Allocation Table
2. File List
3. Disk Data

The files are written in 512-byte blocks. The file system has no directories.

1. File Allocation Table (FAT):

- The table consists of 4096 entries. Each entry is 4 bytes long and bytes are laid out in little-endian order (meaning the most significant byte is at the highest address).
- The first entry must always be 0xFFFFFFFF.
- If an entry is 0x0, then the block is assumed to be empty.
- If an entry is 0xFFFFFFFF, then the block is assumed to be the last block of a file.
- If an entry has a value between 0x0 and 0x1000, the value points to the entry in the table where the next block of the file is located.

Example: Consider three files; “File A” occupies 4 blocks, “File B” occupies 2, and “File C” occupies 1. Then, the following FAT is possible:

Entry	Value	Entry	Value	Entry	Value	Entry	Value
0000	0 x FF FF FF FF	**0001**	0 x 02 00 00 00	**0002**	0 x 05 00 00 00	**0003**	0 x 04 00 00 00
0004	0 x FF FF FF FF	**0005**	0 x 06 00 00 00	**0006**	0 x FF FF FF FF	**0007**	0 x 00 00 00 00
0008	0 x 00 00 00 00	**0009**	0 x FF FF FF FF	**0010**	0 x 00 00 00 00	**0011**	0 x 00 00 00 00
4092	0 x 00 00 00 00	**4093**	0 x 00 00 00 00	**4094**	0 x 00 00 00 00	**4095**	0 x 00 00 00 00

In this table, "File A" occupies the blocks 1, 2, 5 and 6, while "File B" occupies the blocks 3 and 4, and “File C” occupies only block 9. Notice that the entries are laid out in little-endian style, so 0x05000000 simply means the number 5.

2. File List:

- The file list contains 128 items, each of which is 256 bytes long.
- A file item in the list has the following layout:
  - File name: Maximum of 247 characters + the ‘/0’ string delimiter (248 bytes)
  - First block: Location of the first block of the file in the FAT (4 bytes)
  - File size: Size of the file in bytes (4 bytes)

Example: Following the previous example, let “File A” be 2000 bytes long (4 blocks), “File B” be 900 bytes long (2 blocks), and “File C” be 100 bytes long (1 block). Then, the following file list is valid:

Item	File name	First block	File size
000	“File A”	1	2000
001	“File B”	3	900
002	“File C”	9	100
003	NULL	0	0
127	NULL	0	0

3. Data:

File contents are written in 512-byte chunks.

Core Features:

The driver is expected to support the following features:

1. Formatting: Overwrites the disk header with an empty FAT and an empty file list. The user should be able to format a disk from the terminal by typing **./myfs disk -format**

2. Writing: Copies a file to the disk. The command **./myfs disk -write source_file destination_file** should obtain the source_file and write it to the disk with name destination_file.

3. Reading: Copies a file from the disk. The command **./myfs disk -read source_file destination_file** should get the source_file in the disk and copy it to the computer as destination_file.

4. Deleting: Deletes a file in the disk. Command: **./myfs disk -delete file**

5. Listing: Prints all visible files (i.e. files whose names do not start with a ‘.’) and their respective sizes in the disk. Alphabetical order is not required. Command: **./myfs disk -list**

Extra Features:

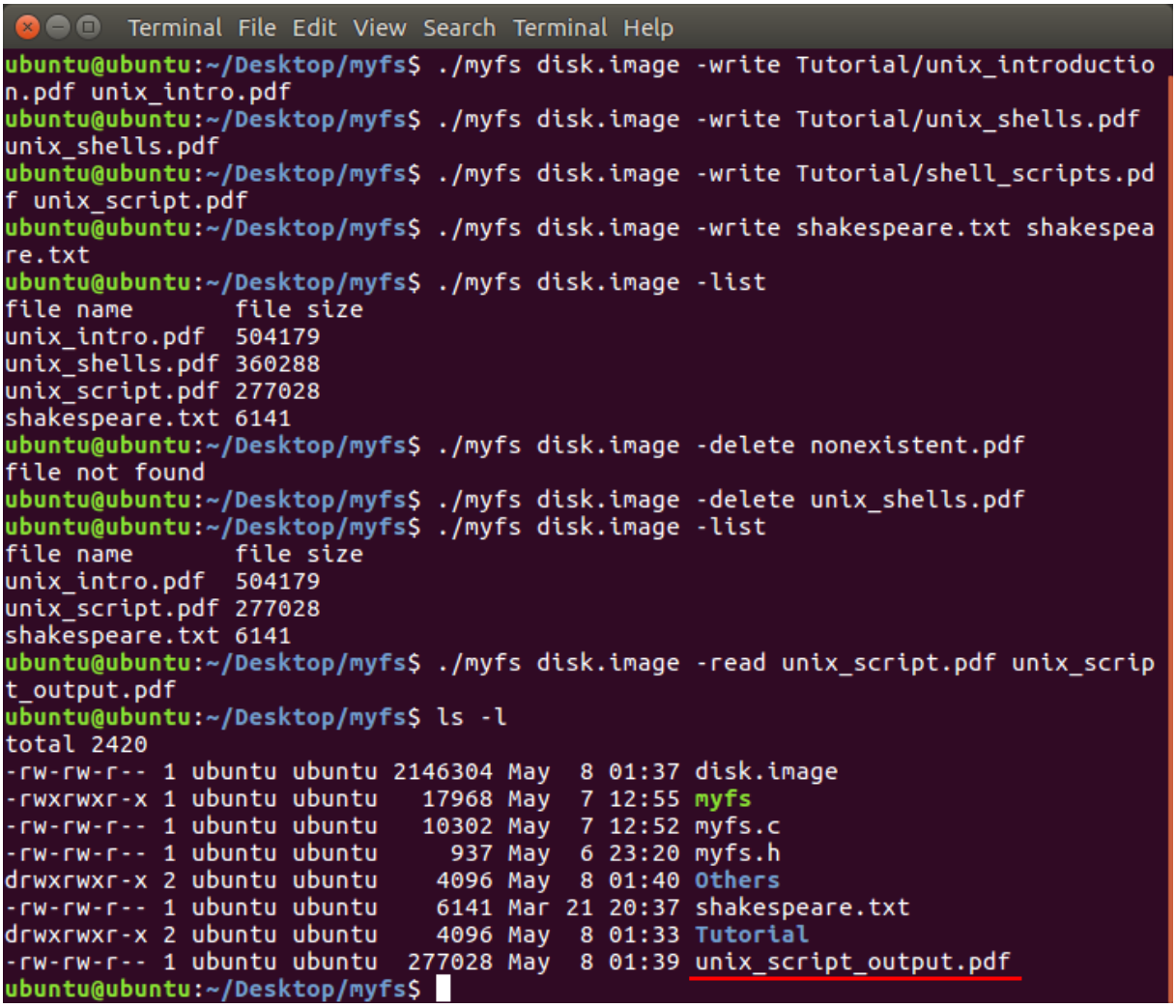
In addition to the five core features, you are expected to implement one of the extra features below.

1. Renaming: Changes the name of the target file in the disk. Sample command: **./myfs disk -rename file_name new_name**

2. Duplicating: Creates a copy of a file in the disk with a new name. Sample command: **./myfs disk -duplicate file_name new_name**

3. Hiding & Unhiding: Hides or unhides the file in the disk. It should throw an error if the user tries hiding an already hidden file or unhiding an already visible file. Sample commands: **./myfs disk -hide file** and **./myfs disk -unhide file**

Example output:

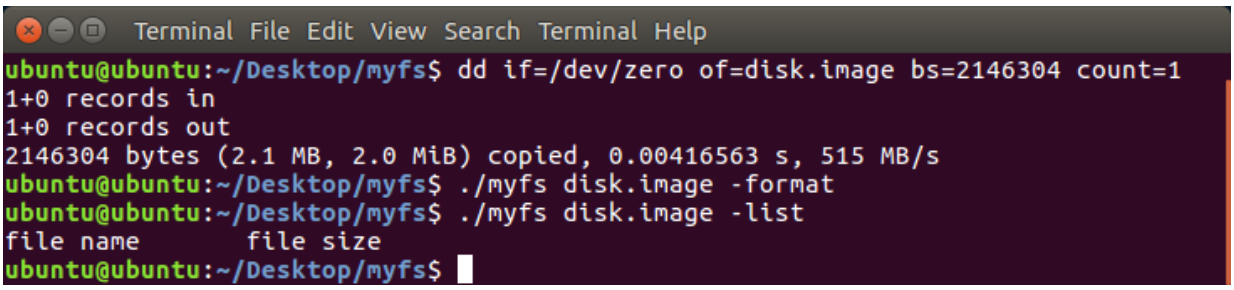


Other Specifications:

Your code should be written in C and compiled with GCC (GNU Compiler Collection). You can compile your code in the command line by typing **gcc myfs.c -o myfs -lm**

Hints:

- For **disk** in the terminal commands, you can use either a real drive or a disk image, since there is not any difference between the two in Unix systems.
  - In order to use a disk image, the image should pre-exist, and should be formatted before used. To create a disk image, simply enter **dd if=/dev/zero of=disk.image bs=2146304 count=1** in the terminal. This will create a "disk.image" file in the current directory. Then, you can format the image, for example, with the command **./myfs disk.image -format**



- Be careful not to use a real drive before backing it up.
- While implementing the features for the driver, you may use the following function prototypes if you like.
  - `void Format();`
  - `void Write(char *srcPath, char *destFileName);`
  - `void Read(char *srcFileName, char *destPath);`
  - `void Delete(char *filename);`
  - `void List();`
- While implementing the writing feature, you can search the file allocation table for empty blocks. For each block, use a buffer to transfer data from the local file to the disk. Do not forget to keep the first block information since you will need to store it in the file list.
  - The buffer can be declared as `char buffer[512];`
  - Before writing the file data to the disk, you can use the `fseek` function to adjust the offset of the file position.
- While implementing the reading feature, you can search the file list for the target file name and pick the first entry whose name matches. Then, you can copy the file block by block using a similar buffer.

Remarks:

1. You should insert comments to your code at appropriate places without including any unnecessary detail. Comments will be graded. You have to write to-the-point comments in your code, otherwise it would be very difficult to understand. If your output is wrong, the only way we can grade your homework is through your comments.
2. Send your homework compressed in an archive file with the name "e<student_ID>_HW4" (e.g. e1234567_HW4.tar.gz). The archive file should include your **source file(s)** and **header file(s)** (if they exist).
3. Your work will be graded on its correctness, efficiency, and clarity as a whole.
4. Late submissions are welcome, but penalized according to the following policy:
   - 1 day late submission: HW will be evaluated out of 70.
   - 2 days late submission: HW will be evaluated out of 50.
   - 3 days late submission: HW will be evaluated out of 30.
   - 4 or more days late submission: HW will not be evaluated.

Good Luck!