Assignment 3: Filesystem

Date: April 30th, 2021
Deadline: May 23th, 2021 23:59 CET

You must implement a tool in Python 3 that understands the Minix filesystem, version 1.

You can use `make tarball` to generate a tarball for submission automatically. This assignment is individual; you are not allowed to work in teams. Submissions are made to Canvas, where they will be tested automatically.

Getting started

1. Read the *Extended Introduction to the Minix Filesystem* provided on Canvas.
2. Unpack the provided source code archive.
3. Familiarize yourself with the Python 3 template `mfstool.py` that we provide.
4. Make sure the `mkfs.minix` and `fsck.minix` tools are installed (e.g. `apt install util-linux` on Ubuntu)
5. Create a test Minix version 1 filesystem supporting 14 character filenames by entering the following commands in a shell:

   ```
   dd if=/dev/zero of=disk1.img bs=1k count=1024
   mkfs.minix -1 -n 14 disk1.img
   ```

   Note that this creates a file first (with `dd`), before it writes a Minix filesystem into that file. After that you can mount the filesystem and add files and directories for testing:

   ```
   sudo mount -o loop disk1.img /mnt
   touch /mnt/file0.txt
   mkdir /mnt/dir1
   ```

   Finally, do not forget to umount, to make sure all changes are written to the disk image:

   ```
   sudo umount /mnt
   ```
6. Familiarize yourself with the Python 3 `struct` module and its `unpack` and `pack` functions.
7. Familiarize yourself with the `bytearray` data type.
8. Familiarize yourself with the Python 3 functions to open, read, seek within, and write files.

Features and grading

Your grade will be 0 if you did not submit your work on time, in an invalid format, use source code from the Internet, or it fails during testing. If your submission is valid, your grade starts from 0, and the following tests determine your grade :

- +1pt if your tool implements the following functionality: Running the command

  ```
  python3 mfstool.py disk1.img ls
  ```

  prints the filenames in the root directory of the `disk1.img` filesystem on standard output (stdout), 1 line per filename, excluding any trailing '\0' characters (see code below). They must be printed in the

order in which they appear in the directory block. It must also print the `.` and `..` directory names. The `disk1.img` file contains a Minix v1 filesystem with 14 character filenames. You may assume all names in this assignment consist of printable characters. Any errors or debugging output must be printed to standard error (stderr). You may assume the root directory data does not occupy more than 1 block.

```
printname = name.rstrip(b'\0')
sys.stdout.buffer.write(printname)
sys.stdout.buffer.write(b'\n')
```

- +1pt if your tool implements the same functionality as above, but now assume `disk1.img` contains a Minix v1 filesystem with 14 character filenames, and the root directory data does not occupy more than 7 blocks. The directory blocks must be processed in the order in which they are recorded in the root directory's inode.
- +1pt if your tool implements the same functionality as above, but now assume `disk1.img` contains a Minix v1 filesystem with 30 character filenames, and the root directory data does not occupy more than 1 block.
- +1pt if your tool implements the same functionality as above, but now assume `disk1.img` contains a Minix v1 filesystem with 30 character filenames, and the root directory data does not occupy more than 7 blocks.
- +1pt if your tool implements the following functionality: Running the command

```
python3 mfstool.py disk1.img cat dirname1/filename1
```

writes the contents of the file `filename1` located in the subdirectory `dirname1` in the root directory of the `disk1.img` filesystem to standard output (stdout). Your tool should support both maximun filename lengths (14 and 30). This holds for all features in this assignment. You do not have to interpret or handle the bytes in the content, just write them to stdout verbatim. The command must not print any extra information, such as extra newlines, on stdout. Hint: `sys.stdout.buffer.write()`. You may assume the file data does not occupy more than 1 block.

- +1pt if your tool implements the same functionality as above, but now assume the file data does not occupy more than 7 blocks.

The above features will get you a sufficient grade for this assignment. So we strongly suggest you focus your efforts on making these work first. You can increase your grade by implementing the below features:

- +1pt if your tool implements the following functionality: Running the command

```
python3 mfstool.py disk1.img touch filename1
```

creates a new file with size 0 and filename `filename1` inside the root directory of the `disk1.img` filesystem. You may assume the root directory data does not occupy more than 1 block, and has space for this filename. For the i-node values of `i_uid`, `i_gid`, etc. see **Remarks**.

- +1pt if your tool implements the following functionality: Running the command

```
python3 mfstool.py disk1.img mkdir dirname1
```

creates a new directory called `dirname1` without any files inside it the root directory of the `disk1.img` filesystem. You may assume the root directory data does not occupy more than 1 block, and has space for this filename. Note an empty directory still needs a `.` and `..` entry. You may assume the disk is not completely full. For the i-node values of `i_uid`, `i_gid`, etc. see **Remarks**.

- +1pt if your tool implements the following functionality: Running the command

```
python3 mfstool.py disk1.img append dirname1/filename1 Hello
```

  appends the bytes "Hello" to the existing file in location `dirname1/filename1` of the `disk1.img` filesystem. Appending means that the bytes are added after the existing content. You may assume this content is smaller than 7168-len("Hello") bytes, and the bytes to append consist of a single word.

- +1pt if your tool implements the `append` functionality but now for files of any size.

The grade will be maximized at 10, so you do not need to implement all features to get a top grade.

> ***Note***
>
> Your tool will be evaluated largely automatically. This means features only get a positive grade if they work perfectly, and there will be no half grade for "effort".

Remarks

- Your tool must use Python 3. Using Python 2 is not allowed.
- Your tool must not use any external commands, non-standard Python libraries, or source code found on the Internet, or you will receive a 0 grade.
- If you use more Python files than `mfstool.py`, you must sure they are included in the submission tar-ball.
- Your tool must support Minix version 1 filesystems, with both 14 and 30 character filenames, for all features you implement.
- Errors or debugging output must go to stderr.
- You can use the `fsck.minix` command to test the integrity of the filesystem you created. You will probably need to use the `-f` flag to force it to check a filesystem that is marked "clean".
- In the i-nodes you create, you may set the `i_uid` and `i_gid` to 0, `i_time` to current time, and `i_nlinks` field to 1. The `i_mode` field should correctly reflect whether the i-node is about a directory (`S_IFDIR`) or a regular file (`S_IFREG`). The permission bits of the `i_mode` fields must be `S_IRUSR | S_IWUSR | S_IXUSR`.
- You do not have to maintain the number of links to i-nodes, so can also ignore warnings about these from `fsck.minix`.
- You may assume that a zone consists of just 1 block.