Operating Systems
Jacobs University Bremen
Romelda Blaceri

Course: CO20-320202
2 December, 2018

OS 2019 Problem Sheet # 6

Problem 6.1

Solution:

a) The directory that is contained in the new file system is a `lost+found` directory. When it needs to repair the file system, `fsck` (filesystem check) might find data fragments that are not referenced anywhere in the filesystem and it turns these almost-deleted files back into files. The idea is that the file had a name and a location once, but since that information is not available anymore, `fsck` deposits the file in a specific directory, called `lost+found`.
Files that appear in `lost+found` are typically files whose name has been erased (they were unlinked) but the data wasn't erased yet, so they were still opened by some process when the system halted suddenly. Files which are lost because of corruption would be linked to the corresponding file system's `lost+found` inode.

b) Both free blocks and available blocks are provided to be used by the user of the file system. The difference is that there are more free blocks in quantity than available blocks. It is not possible to add more data in the file system after the user uses all the available blocks, even though there may be many free blocks left. These free blocks are reserved for the root of the file system.

c) When we delete the underlying file `vhd.ext3`, nothing actually happens to the mounted file system. This is because even though `vhd.ext3` is unlinked, the memory occupied by the mounted file system remains in the memory until the file system is umounted.

d) The created file is nearly 4 megabytes. The free blocks number doesn't change, while the free inodes number decreases by 1. This happens because one inode is allocated in order to store the metadata of `big.data`.

e) The command `chattr +i` adds an attribute to `big.data`, where `+i` attribute makes the file 'immutable', therefore the file cannot be deleted, cannot be modified (we cannot write data to it, we cannot rename it), and cannot be linked. For this reason, the `rm` command fails. By performing `lsattr big.data` we can list the attributes of our file.

f) When the `chroot` command was executed, the root of the file system was changed such that it became `mnt`, so `mnt` was behaving as `/`. It was important to copy a statically linked version of `busybox`, because other libraries were prevented to get linked. Therefore, some shell commands could not be used anymore since their libraries didn't exist in the new file system.

Problem 6.2

Solution:

a) The workdir option is used to prepare files before they are switched to the overlay destination (upperdir) in an atomic action. Therefore, `top` is non-atomically created and configured in workdir (`over`) and then atomically moved into upperdir (`upper`). For the second case again, since upperdir is the directory with which lowerdir is overlaid with, if duplicate filenames exist in lowerdir and upperdir, upperdir's version takes precedence even after we create `lower/low`. If we append data to `over/low`, the file will be changed in `upper/low` and `over/low`, but not in `lower/low`. After we unlink `over/low`, we can notice that the file will remain in `lower`, it will be removed from `over`, and it will appear as a pipe in `upper`. The file system remembers that `over/low` got unlinked because a `whiteout` file is created in `upper`. If we change the permissions of `over/lo`, the changes will be reflected also in `upper/lo`, but not in `lower/lo`.

b) Overlay is great for systems that rely on having read-only data, but the view needs to be editable, such as Live CD's and Docker containers/images (image is read only). Since it also allows us to quickly add some storage to an existing file system that is running out of space, without having to alter any structures, it could also be a useful component of a backup/snapshot system.

c) Concerning the copy_up process, it first makes sure that the containing directory exists in the upper file system, then creates the object with the same metadata and then if the object is a file, the data is copied from the lower to the upper file system. Finally any extended attributes are copied up, and once the copy_up is complete, the overlay file system simply provides direct access to the newly created file in the upper file system - future operations on the file are barely noticed by the overlay file system (though an operation on the name of the file such as rename or unlink will of course be noticed and handled).
As for the multiple lower layers, they can be formed in the following way: e.g.
`mount -t overlay overlay -olowerdir=/low1:/low2:/low3 /over`
Notice that `upperdir=` and `workdir=` may be omitted, and the overlay would then be read-only. The specified lower directories will be stacked beginning from the rightmost one and going to the left, so in our example we would have: low1 will be the top layer, low2 the middle one and low3 the bottom one.

References:
https://askubuntu.com/questions/109413/how-do-i-use-overlayfs
https://www.kernel.org/doc/Documentation/filesystems/overlayfs.txt
https://blog.programster.org/overlayfs
https://docs.docker.com/storage/storagedriver/overlayfs-driver/