Operating Systems
Jacobs University Bremen
Jovan Shandro

Course: CO20-320202
02-12-2019

Assignment 6

Problem 6.1

Solution:

a) The directory that is contained in the new file system is called 'lost+found'. Inside it, recovered bits of corrupted files can be found. This directory is used by the file system check command (fsck) to identify inodes that cannot be accessed by any directory (which are basically lost). These could be files, directories, and even devices. Recovering is a bit tough, as if something is in the lost+found folder, it usually won't be the entire file. Instead, you'll see small pieces of files or bits of corrupted data, and there is not much you can do with them.

b) After running **stat -f .** I get that there are 2905 free blocks and 2701 available ones. Notice that there are more free blocks than available blocks. They are both for the user of the file system to use, but the reason why we see the change in numbers is that those free blocs are reserved for the root of the file system, so if the user uses all the available blocks, he/she/it can't add any more data to the file system, even though there are many free blocks left (used for what I wrote above).

c)After deleting the file vhd.ext3, nothing happens, because we are not actually deleting it completely from memory, we are just unlinking it. This means that even though we removed the file, it is still in the memory of the mounted file system and it will only be removed once we unmount (or umounted as that is the actual command) it.

d) The created file has a size of 4,194,304 bytes. Running **stat -f .** again, I noticed the number of free and available blocks have not changed. The only change that happens is that the number of free inodes is reduced by one (in my case from 1013 to 1012). The reason for this change is that this inode has been used to store the metadata of the newly created file 'big.data'.

e) Using the command **sudo chattr +i big.data**, what basically happens is that an **immutable** attribute is added to the big.data file. This means that the content of the file cannot be modified, renamed, deleted etc, and thus the second command **sudo rm big.data** fails resulting in "rm: cannot remove 'big.data': Operation not permitted". To display the file attributes lsattr can be used and it results in printing "—-i————— big.data " indicating that the file has a +i attribute.

f) i) Executing the **chroot** command changes the root of the file system. So mnt now becomes the root of the file system.

ii) It is important to use the statically linked version of busybox to prevent the dynamic linking of other libraries. This fact explains why many shell commands do not work in the file system (e.g cat)

Problem 6.2

Solution:

a) The file over/top is actually stored in the upper directory. (References: 1, 2, 3).
Appending data to over/low, we notice that the changes are not reflected in lower/low, but only in over/low and upper/low (because as stated in one of the references any change in over will be reflected in the upper directory.). This also happens as the files in the lower directory will be read only. So the actual data is stored in the over directory and the change is also reflected in the upper directory. Another way to see this is, as the exercise continues, to try to unlink over/low. After doing that we see that lower/low is untouched, however in upper we see a pipe named low (which was used to reflect the changes). However if we would unlink upper/low there would be no change in over/low which means that the actual data is stored in the over directory. The file system remembers that over/low got unlinked because a *whiteout* file is created in the upper directory. Then we create lower/lo and change the permissions of over/lo. We notice that the permissions change for over/lo and for upper/lo, but there is not change in the permissions of lower/lo.

b) According to ”Click me!”, two of the cases in which overlay file systems are used are:
i) For Live CD creation, where a read-only image is augmented with a writable layer in tmpfs, thereby enabling a dynamic, but ephemeral session.
ii) It is used by the well known platform 'Docker' for its containers for the same reason.

c) As also mentioned in the references above, yes it is possible to stack multiple lower layers. For example the following:
mount -t overlay overlay -olowerdir=/lower1:/lower2:/lower3,upperdir=upper,workdir = merged
creates 3 lower layers that are stacked s.t. lower1 is on top of lower2 which is on top of layer 3. Upper is on top of lower1. As for the coping of metadata, overlay file systems will only copy up the metadata (not to whole file!!!!!), when a metadata specific operation like chmod or chown is performed, whereas the full file will be copied up later only when the file is opened for write operation.