Operating Systems
Jacobs University Bremen
Arsenij Percov

Course: CO20-320202
December 1st, 2019

Homework 6

Problem 6.1

Solution:

a)
By default ext3 creates folder lost+found, and reserves space for it. That is done for fsck tool. fsck checks and repairs the filesystem, and in case it finds a data that looks like a file, but doesn't have name and is unlinked, but still has inode, then fsck restores the file and stores it in the lost+found folder.
Reference: `https://unix.stackexchange.com/questions/18154/what-is-the-purpose-of-the-lo`

b)
Free space, is amount of space not used totally.
Available space, is amount of space not used and not reserved.
Shortly : available space = free space - reserved filesystem blocks.
Some portion of memory is reserved to be used by root only.
Reference: `https://serverfault.com/questions/228514/free-disk-vs-available-disk`

c)
Nothing happens. We didn't delete it, but unlinked, the data is still saved, and will be protected, as long it used in this case it is mounted, so we will have to unmount it first.

d)
The size of the file is 4194304 bytes.
The free blocks number didn't change. The number of inodes decreased by 1.
The behavior can be explained by the fact, that only inode was created with metadata depicturing the size. We haven't written a single byte to the file, but the size is explained by using seek parameter, that skipped N blocks. Block's size was defined by the bs parameter, in our case block size is 1kb, and seek is set up to 4096. That skips in total 4096kb of data, which is equal to the size of the file stated in inode structure - 4194304 bytes.
However, that memory was not even allocated, it is technically a hole. The start of the file is moved by that offset by seek() systemcall, so no data is written there ever. Therefore, they are no allocated by the filesystem, and stats of filesystem do no use that size in calculations of free and available size.
Reference: `https://unix.stackexchange.com/questions/108858/seek-argument-in-command-dd`

e)
With the first command we added attribute to the big.data file. To be more specific we added "immutable" attribute. It means that the file will be immune to any change in the future, so deleting, writing, changing it metadata, or even name will not be possible. Because of that when we tried to delete the file, we recieved "rm: cannot remove 'big.data': Operation not permitted" error.

f)
chroot command changed the root to location mnt, that means that whenever we would call'/' that would resolve to mnt directory, since it is the new root. However, the purpose of chroot is a bit different, it executes the command provided as second argument to its call, with root directory set to provided new root. That means that it will call /bin/sh, and '/' will be mnt. /bin/sh is link created by previous command ln, and it resolves to 'mnt/bin/busybox'. So the final result is busybox executed in sandbox, where root is set to be mnt.
The program has to be linked statically, since otherwise it will try to link the file, following link that were created outside of the sandbox, and that had different roots, so they will not be accessible and will lead to wrong probably non-exiting directories.

Problem 6.2

Solution:

a)
The file called top is stored in upper directory, since that one is meant to be write directory.
The file called low is actually created in lower directory, since it is accessed directly, and not

through over directory.
If you unlink over/low that proccess will be simulated by creating whiteout file, it only exist in overlay filesystem, and doesn't physically exists in directories. When we unmount the system, the changes indicated by whiteout files will be applied to lower directory.
When we create file lo in the lower directory, and try to change its permission, the file is copied from lower to upper dir, and its permissions are changed only at one that is at upper, however, that means that any time we access that file through over, we will access the copy with modified permissions.

b)
One of usage are live cd, where you have read-only image, and then you have writeable layers overlaying it, where you can create dynamic sessions, and allow changes that will not affect the image.
Second usage is for creation of filesystems for Linux container used by environment like Docker.
Reference: `https://windsock.io/the-overlay-filesystem/`

c)
No. If metadata only copy up feature is enabled, then only metadata will be copied up to the tree, as operation that doesn't affect data of file is called. To turn on the feature we need to enable CONFIG_OVERLAY_FS_METACOPY config option.
Reference: `https://www.kernel.org/doc/Documentation/filesystems/overlayfs.txt`

Yes, it is possible to stack multiple layers. You provide list of directories in top-to-bottom order separated by ':'. All changes will displayed in upper directory.