Operating Systems
HW Sheet # 6

Submitted by: Neeha Hammad
Date: 2019-12-02

Problem 6.1

a) There's a lost+found directory present within our new file system. This usually contains files that are unlinked but might be opened in other programs. From here, files can be recovered.

   lost+found comes in handy in case our file system goes through any corruption. In such a case, fsck, which is used to check and repair a file system is used. Fsck uses lost+found in a way that all files that would normally be lost because of directory corruption get linked in that filesystem's lost+found directory via the inode number.

   Source:https://unix.stackexchange.com/questions/18154/what-is-the-purpose-of-the-lostfound-folder-in-linux-and-unix

b) In this particular case, we see that we have 2905 free blocks but only 2701 blocks are available. Free blocks, which have a larger quantity are all unused blocks. Available blocks are all those free blocks which can be used by the user. The remaining free blocks (eg: 204 here) are reserved for the root of the file system. The system needs some of this free space to run properly.

   Source: https://serverfault.com/questions/228514/free-disk-vs-available-disk

c) Nothing happens if we remove that file. If we read remove()'s documentation, we see that calling remove calls unlik() when used for files. So basically, we only unlink vhd.ext3 from our file system. Unless we "unmount", the memory within our new mounted filesystem will persist.

d) If we check by running ls -s or ls -sh, we see that the size of big.data is just 0.

   The number of free inodes have decremented by 1. This is because 1 inode allocated to store big.data's metadata. Number of free blocks remain the same.

e) The chattr command is used to change file attributes on a Linux file system. The i option used with chattr stands for "immutable". Thus, the chattr +i command makes big.data immutable. This means that even a root user will be prohibited from deleting the file.

Additionally, now big.data file or even its name cannot be modified and no new links can be created to it.

The lsattr command can be used to display file attributes. To view files with the immutable attribute in specific, we can use such a command:

lsattr -R | grep +i

Sources:https://unix.stackexchange.com/questions/132795/how-to-search-for-files-with-immutable-attribute-set

http://www.aboutlinux.info/2005/11/make-your-files-immutable-which-even.html

f) By using chroot, we change the apparent root directory of our file system from / to mnt. Now, mnt is the "root" for our file system.

If we don't use a statically linked version, other libraries will also be linked for usage. As observed, in the current situation, we are restricted to using limited shell commands as for others, the libraries are not present within the file system that we created.

Problem 6.2:

a) 1. Upon touch over/top, the file top is created in the folder over as well as upper. It's stored in upper.

2. Upon creating lower/low, the file low is created in over and lower but when we append data to over/low, the data is actually stored in upper. The file low starts appearing in upper too. The data is written in the low file from the over folder and upper folder but not in the low file from the lower folder.

3. Basically, linking makes the file anonymous and makes it a candidate for deletion. The file is deleted in case there are no other processes which have the file opened. When we unlink over/low, the low file gets deleted from over. The low file still exists in upper but it cannot be accessed anymore. The linking info is also stored in upper.

4. If I use chmod 777 over/low, the file becomes readable, writable and executable by everyone.

/* In overlying file systems, modifications to files in the "upper" directory take place as usual. However, any modification to a file from the "lower" folder creates a copy in the upper directory, and that file will be the one modified. This leaves the base files untouched and available through direct access to the "lower" folder. */

Source:https://www.datalight.com/blog/2016/01/27/explaining-overlayfs-%E2%80%93-what-it-does-and-how-it-works/

b) Live CDs, which can be used for a quick demo or test of Linux (eg Ubuntu) are a great example. A live CD is a complete bootable computer installation which can run directly from a CD-ROM into a computer's memory, instead of loading from a hard drive. By using a Overlay FS, one (usually read-write), directory tree is allowed to be overlaid onto another, read-only directory tree. All modifications go to the upper, writable layer. Slackware uses Overlayfs for their Live CDs.

   We boot read-only image and combine it with “ram disk” using Overlay FS. All the changes are written to the ram disk. After rebooting, everything loads to the previous state. For example, Internet Kiosk.

   Additionally, each user can have a personal thumbdrive. Instead of ramdisk, we can use a thumb drive. We can retrieve data from here to “return to previous state” (during next Live CD’s boot). Next, we can commit changes before shutting down the system.

   Another use is dockers.

c) Unless we are mounting the overlay as noatime, no. Othweise, the overlay file system only copies the metadata if the overlay is read/write with access times being updated continuously.

   Yes, multiple lower layers can be stacked using the : symbol. The specified lower directories are stacked from right to left.