CS431 — Exercise 5

February 22, 2017

Due: Monday, February 27, 2017 before midnight (50 points)

In this exercise you will learn about creating and working with file systems in a Linux environment. You can work with a partner for this exercise. If you do, only one partner should create the repository and add the other with at least developer privilege. Make sure both partners have their names on the submitted documents.

For all of the steps below, document the results of performing the various actions including screenshots. Instead of code, submit a document containing your comments and screenshots.

Virtual Machine

I recommend using a virtual machine to complete this exercise so you do not risk damaging the file systems on your main computer. You can use VirtualBox or VMWare to do this, I recommend VirtualBox for people with less experience using virtual machines. I won't leave a lot of instruction for setting up a VM in this document but please see me for assistance if you have not done this before. I will assume you are using VirtualBox for the rest of this document.

You should install a Linux distribution (I recommend Xubuntu for those familiar with Windows) to the virtual machine. To do this, you should download the OS installer disk image (usually a `.iso` file) then boot from that disk image in your virtual machine.

Adding a Disk

Once you have created your virtual machine and installed the Linux distribution, you should add a new virtual disk. First, shut down the virtual machine. In VirtualBox, a disk can be added by right-clicking the VM, going to Settings, then Storage. You should see a list labelled "Controller: SATA". To the right is a small hard drive icon with a green plus sign. Press that to add a new disk. You should create a new disk. All of the default settings are fine, but you may want to use a smaller size as this is just for testing purposes. I made a disk of just over 100MB.

Creating Partitions

Once a virtual disk has been added, boot the virtual machine up then open a terminal and type

```
$ ls /dev/sd*
```

You should see something like this as the result:

```
$ ls /dev/sd*
/dev/sda /dev/sda1 /dev/sda2 /dev/sda5 /dev/sdb
```

In Linux, each disk is given a label `sdX` where `X` is a unique letter. Each partition on disk `X` is then numbered from 1, so `sda1` is partition 1 on disk `sda`. Special files exist in the `dev` folder that correspond to each of these disks/partitions. If you wanted to write random data over an entire disk you could do so by

```
$ dd if=/dev/urandom of=/dev/sdX
```

where `X` is the disk to write. This is a useful tool, but can be dangerous if misused.

In our case, the original Linux distribution was likely installed on various partitions on `sda`. The new disk we created is therefore `sdb`. Since this disk is new with no partitions yet, we have no `sdb1`, `sdb2`, etc. Before we can create a usable file system on this disk, we have to create various partitions. This involves creating then editting the master boot record for the disk.

This can be done with the `fdisk` program. There are alternative programs such as `gparted` that comes with a GUI if you prefer to use them instead of `fdisk`.

Create four partitions on your disk, in my case I made each one 25MB. We will put a different file system on each one. After doing this, running `ls /dev/sd*` again should give us output like this:

```
$ ls /dev/sd*
/dev/sda  /dev/sda2 /dev/sdb  /dev/sdb2 /dev/sdb4
/dev/sda1 /dev/sda5 /dev/sdb1 /dev/sdb3
```

This now shows four partitions for disk `sdb`.

Creating ext4 File System

Use the `mkfs.ext4` command to create a new `ext4` file system on `/dev/sdb1`. Once created, we can *mount* the file system in to our operating system's directory tree so that it can be accessed. First, make a directory to mount this file system to such as `/mnt/ext4`. Next, use the `mount` command to mount the file system. Once mounted, change directory in to the new file system and create some files. After this, use the `umount` command to *dismount* the file system.

Creating Swap Space

For the next partition, we will be converting it to swap space. Swap space is the space on disk used as virtual memory by the Linux operating system. First, run the following command:

```
$ free -m
```

This will show free memory including swap space. In my output, it says the system has 509MB of total swap space. Check the value for your system. Next, use `mkswap` to create the swap space on `/dev/sdb2`. After creating it, use `swapon` to enable the swap space. Finally, run `free -m` again to check that the space is now in use by the system. In my case, it now says the system has 534MB of total swap space.

Now, reboot the system and run `free -m` again. You will notice the new swap space is gone. `swapon` is a temporary command. Once rebooted, the system does not remember the added swap space. We can fix this by editting the *file system table file* which is normally `/etc/fstab`. This file describes which file systems to mount while booting the operating system.

Add a line to the bottom of the file like this:

```
/dev/sdb2   swap    swap    defaults    0   0
```

Look through the documentation for `fstab` and describe what each of these values refers to. After modifying the file, reboot the system and run `free -m` again to check the swap space in use. You should again see the extra swap space from our `sdb` disk.

We can also use `fstab` to create what is known as a *RAM disk*. This is a file system that exists entirely in memory for high performance, temporary file storage. Linux has a file system type known as `tmpfs` that will act as such a file system. Try adding the following to `/etc/fstab`:

```
temp    /mnt/temp   tmpfs   defaults,size=10M   0   0
```

This will create a RAM disk of size 10MB and mount it at `/mnt/temp` when you reboot the system. I use this technique to create a "scratch" folder where I put various files that I don't need to keep when I reboot my machine. This gives the benefits of auto-cleanup every time I reboot and the high performance of using only RAM.

Creating a RAID1 Mirror

For `sdb3` and `sdb4`, we will use `mdadm` to create a RAID1 mirror. This creates an abstraction layer providing us with a new device that is the result of performing the RAID while `mdadm` utilizes each disk. Use `/dev/md0` for your new RAID1 mirror when entering the `mdadm` command. Once you have successfully created the RAID1 mirror with `mdadm`, run the following command:

```
$ cat /proc/mdstat
```

This will print out the status of your RAID devices. For example, with the new RAID1 mirror I see this output:

```
$ cat /proc/mdstat
Personalities: [raid1]
md0 : active raid1 sdb4[1] sdb3[0]
      25536 blocks super 1.2 [2/2] [UU]

unused devices: <none>
```

At this point, we have not created a file system on `sdb3` or `sdb4`. Instead, the RAID is an intermediary between the partitions and a file system we can put on top of them using the new device `md0`.

Now, let's create an encrypted file system on top of this RAID1 device. To do this, we will use LUKS (Linux Unified Key Setup). The program to create and manage encrypted file systems is `cryptsetup`.

Use `cryptsetup luksFormat` to create an encrypted file system on `/dev/md0`. When creating an encrypted file system, you must provide a passphrase that will be used to decrypt it later. You can then use `cryptsetup luksOpen` to open the encrypted file system. This will put a reference to it in `/dev/mapper/`. For example, if I open it has tempcrypt then I would have access to the encrypted disk by `/dev/mapper/tempcrypt`. You can then use `mkfs.ext4` to create a file system on `/dev/mapper/tempcrypt`. You can finally mount the newly created file system using the `mount` command. Any files that you create in this new file system will be encrypted automatically and also mirrored between `sdb3` and `sdb4`.

Once you have created and mounted the encrypted file system on top of the RAID1 mirror, make some files, dismount it, and then close it using `cryptsetup luksClose`. Make sure to document these actions.

Submission

1. On https://codebank.xyz, create a project named `CS431-EX5`. Follow this naming convention precisely including case.
2. Create your own local repository by the following:
   (a) Navigate to your local directory for this project.
   (b) Run `git init` to initialize the repository.
3. Make sure you add the reference to the remote repository in your local repository with:

```
$ git remote add origin https://codebank.xyz/username/CS431-EX5.git
```