IT1100 : Introduction to Operating Systems

Chapter 15

What is a partition?

A partition is just a logical division of your hard drive. This is done to put data in different locations for flexibility, scalability, ease of administration, and a variety of other reasons.

One reason might be so you can install Linux and Windows side-by-side.

What is a partition?

Another reason is to encapsulate your data.

- Keeping your system files and user files separate can protect one or the otherfrom malware. Since file system corruption is local to a partition, you stand to lose only some of your data if an accident occurs.
- Upgrading and/or reformatting is easier when your personal files are stored on a separate partition.
- Limit data growth. Runaway processes or maniacal users can consume so much disk space that the operating system no longer has room on the hard drive for its bookkeeping operations. This will lead to disaster. By segregating space, you ensure that things other than the operating system die when allocated disk space is exhausted.

Linux Partitions

In Linux, a minimum of 1 partition is required for the `/`.

`Mounting` is the action of connecting a filesystem/partition to a particular point in the `/` root filesystem. I.e. When a usb stick is inserted, it is assigned a particular mount point and is available to the filesytem tree. - In windows you might have an A:, or B:, or C:, all of which point to different filesystems.

What is Swap?

If RAM fills up, by running too many processes or a process with a memory leak, new processes will fail if your system doesn't have a way to extend system memory. That's where a swap area comes in. A swap space is a hard disk partition where your computer can "swap out" data from RAM that isn't being used at the moment and then "swap in" the data back to RAM when it is again needed. Although it is better to never exceed your RAM (performance takes a hit when you swap), swapping out is better than having processes just fail.

More on Swap

General rule of thumb is to have your swap space be 2x the size of your RAM (if you have less than 1GB of ram, otherwise the same size as RAM)

Windows and Mac OS have their own form of swap files.

Swap isn't absolutely critical on newer Linux distros but may impact performance if you don't have it.

More on disk partitions

- When a new disk is inserted (mounted), it will show up in the filesystem hierarchy under devices `/dev/`.
  - Most devices including usb drives and SATA drives are treated as a scsi drive and will show up as `/dev/sda` or `/dev/sdb` or some `/dev/sd?` depending on how many devices you have plugged in.

More on disk partitions

- Device naming conventions
  - Each physical device can be partitioned.
    - `/dev/sda1` #first partition on first drive of type scsi
    - `/dev/sdb7` #seventh partition on second drive of type scsi

Partitioning outside the installer

Partitioning can be done during the install and after installing as well.

- Most of the current linux filesystems are `ext4`.
- Volume labels make it possible for partitions to retain a consistent name regardless of where they are connected, and regardless of whatever else is connected. Labels are not mandatory for a linux volume. Each can be a maximum of 16 characters long.
- Why do we care? Sometimes if you remove a device and plug it back in, it could be re-assigned (i.e. the first partition of the second lowest numbered SCSI drive is /dev/sdb1. If the drive referred to as /dev/sda is removed from the chain, then the latter partition is automatically renamed /dev/sda1 at reboot.)

Steps for partitioning

1. Use `cfdisk` or another program to partition free space
2. Run the `mkfs` command to format the partition(i.e. mkfs.ext2 /dev/sdb5)
3. Create a mount point (i.e. `mkdir testmount`)
4. Run the mount command (i.e. `mount /dev/sdb5 testmount`)

These steps are clarified during the assignment.

Making mounts persist

When issuing a mount command at the command line, the mount will only persist until the machine is rebooted. If we always want that particular mount to persist we can put an entry in `/etc/fstab`.

- `cat /etc/fstab`

Ubuntu now uses UUID to identify partitions.

Making mounts persist

Why use a UUID instead of /dev/sda1 to identify the device? Say that that you plugged another disk into your computer and booted up. It probably won't happen, but it is possible that the new disk might be identified as /dev/sda, causing the system to look for the contents of /boot on the first partition of that disk.

This is why the partitions created during installation are identified by a UUID in the /etc/fstab file.

To find out what the UUID is for all devices we can use the block id command:

- `sudo blkid`

Making mounts persist

Here is an example entry using the device name instead of the UUID.

- `/dev/sda6 /home/joe/op auto defaults 0 0`

Here is a short explanation:

- filesystem: `/dev/sda6` - the device we are going to mount
- mount point: `/home/joe/op` - the mount point
- type: `auto` - automatically detect what fs type this is (ext3 or ext4 for example)

- options: `defaults` - options associated with mount
- dump,pass: `0 0` - advanced, usually these are always 0 for added devices.

MSDOS

- Legacy partitioning system (still widely used)
- Each drive or device can have up to 4 primary partitions or 3 primary partitions and 1 extended partition.
- Extended partitions expand the capacity of the devices allowing for more than 4 partitions. An extended partition can have many logical partitions. Often listed as unlimited.
- Primary and the one Extended partitions are always numbered 1-4.
- Logical partitions are always number 5 or higher.

What is the MBR - Old Way

The information about how a hard disk has been partitioned is stored in its first sector (that is, the first sector of the first track on the first disk surface). The first sector is the master boot record (MBR) of the disk; this is the sector that the BIOS reads in and starts when the machine is first booted. The master boot record contains a small program that reads the partition table, checks which partition is active (that is, marked bootable), and reads the first sector of that partition, the partition's boot sector (the MBR is also a boot sector, but it has a special status and therefore a special name). This boot sector contains another small program that reads the first part of the operating system stored on that partition (assuming it is bootable), and then starts it.

What is the GPT - New Way

To see which partitioning scheme we are using use the following command: `sudo parted -l`. If we see the `ms-dos` partition table this is what we are using which is the old MBR method. If you see gpt, you are using the GPT, GUID partition table.

Advantages to the GPT:

- GPT allows for large disks and large partitions, more than 2 TB.
- GPT allows unlimited number of partitions.
- There are other advantages to using GPT for large systems that reduce problems and help in error recovery including GPT data structures are stored twice on the disk.

Creating a filesystem

After we create the partition, we then proceed to put a filesystem on it. This is accomplished with the `mkfs` command (or some derivative). What types are available?

- ntfs
- ext3
- ext4
- btrfs
- vfat
- more...

Inodes

First watch this video.

The filesystem is made up of inodes - a slot that contains metadata about a file. The inode contains the following.

- size
- device id
- UID, GID
- A pointer to the file location on the Hard Drive

- Other info found using `ls -l` and `stat`
  - `ls -lai` will give us the inode number and the metadata.
  - `stat f1.txt` will give us additional metadeta information.
- And a bunch of other cool stuff about the file.
- THE INODE DOES NOT STORE THE FILENAME.

Inodes

`df -i` will show us how many inodes we have and how many are available to us.

- We cannot have more files than inodes.
- It is more common to use all the disk space and still have inodes left.

Viewing inodes

All unique files and directories on the same partition will have a unique inode number

- `ls -lai /`

The first number you see on each line is the inode number. The reason you see multiple “1’s” and “2’s” is because of the system partitions created when the computer starts.

- Use the spaces inside quotes to refine your results
  - `mount | grep -e "sys " -e "proc " -e "dev "`
- root(/) generally has an inode of 2

Viewing inodes

If you find two filenames on the same partition with the same inode number then the files are not actually different files, but the same file with different names. Such as a hardlink link. Once created, a hardlink will always link to the file using the inode number even if you rename, move or delete the original filename.

Symbolic links, however, get a unique inode number and link to the name of the file not the file itself. So if you change the original filename the symbolic link is broken.

Viewing inodes

If they have the same inode number and you know for sure that they are not the same file, then they are on different partitions.

Log into scratch -

- `mkdir inode-fun`
- `cd inode-fun`
- `touch sunshine.txt`
- `touch rainbow.txt`
- `ln -s sunshine.txt inode-symlink`
- `ln rainbow.txt inode-hardlink`
- `ls -lai`
- `ls -lai ../ | grep inode-fun`

Viewing inodes

Examine inode numbers of the files -

- What is the inode number of each file
- Do any of the files have the same inode number?
- If so, why?
- Remember that dot(.) stands for self and dotdot(..) stands for parent directory. They are hard links.

Files can have multiple names. This is why the inode doesn’t store the filename.

- `ls -lai /`
- Look again at dot(.), dotdot(.). These stand for `/` because there is nothing above the root of the computer in the tree. Notice that it is very different than `/root` (the home directory of the root user).

Renaming and Moving Files

A file’s inode number stays the same when we rename the file and when moved to another directory on the same device(partition).

- Remember the inode number for `rainbow.txt` and `sunshine.txt`
- `cd ~/inode-fun`
- `ls -lai`
- `echo "hardlinks are amazing" >> rainbow.txt`
- `cat rainbow.txt`
- `mv rainbow.txt ../`
- `ls -lai ../rainbow.txt`
  - What happened to the inode number?

Renaming and Moving Files

- `ls -lai`
  - What happened to the inode number of inode-hardlink?
- `cat inode-hardlink`
  - Is it still linked? Why or Why not?
- `echo "Inodes are cool!" >> sunshine.txt`
- `cat inode-symlink`
- `mv sunshine.txt sun.txt`
- `cat sun.txt`
- `ls -lai`
  - What happened to the inode number of inode-symlink and sun.txt?
- `cat sun.txt`
- `cat inode-symlink`
  - Is it still linked? Why or Why not?

Renaming and Moving Files

Choose one of the inode numbers and do a search. - `find / -inum 147649`

Where is the NAME of the file?

The filename or path of the file is NOT in the inode. It’s NOT in the data blocks (With the contents of the file on the Hard Drive). It’s in the metadata of the directory.

You see, the metadata of the directory contains a table of the filenames in the directory, and the matching inodes. Think of it as a table, and the first two entries are always “.” and “..” The first points to the inode of the current directory, and the second points to the inode of the parent directory. The remainder of the entries in the table are the directories and files within the current directory.

Where is the NAME of the file?

When you move or rename a file within the same partition, you don’t actually move the data (contents). That would be Slow. You just create the -name,inode- entry in a new directory, and delete the old entry inside the old directory. In other words, moving a gigabyte file takes very little time. In the same way, you can move/rename directories very easily. That’s why “mv /usr /Old_usr” is so fast, even though “/usr” may contain (for example) 57981 files.

When you copy a file - it does create a duplicate of the same data and the copy gets its own inode number.

To copy a gigabyte file or directory takes time because every byte must be copied.

Where is the NAME of the file?

When you create a hard link, it just creates a new name in the table linked to an already existing inode number without moving or copying the file. So if you move, rename or delete one of the hardlink names - all other names linked to that inode still work and the data is still accessible.

Symbolic links break when you move, rename, or delete the file that it links to because they are linked to the filename not the inode number.

Where is the NAME of the file?

- `ln -s sun.txt sun-symlink`
- `cat sun-symlink`
- `cp sun.txt rain.txt`
- `echo "This file is a copy" >> rain.txt`
- `cat rain.txt`
- `cp rain.txt wind.txt`
- `cat sun.txt`
- `ls -lai`
    - What is the inode number of rain.txt? Is it linked to sun.txt or just a copy of the file?

Where is the NAME of the file?

- `ln rain.txt rain-hardlink`
- `echo "Now it has a hard-link too" >> rain.txt`
- `cat rain-hardlink`
- `ls -lai`
    - What is the inode number of rain-hardlink? Is it linked to rain.txt or just a copy of the file?
- `rm sun.txt`
- `cat sun-symlink`
    - Why does it not work anymore?
    - What will happen to rain-hardlink if we delete rain.txt?
- `rm rain.txt`
- `ls -lai`
- `cat rain-hardlink`
    - Why does it still work when the symlink did not?

Inode Metadata

Run the command `ls -la ~/inode-fun`. What is the field after the permissions? This is count of hard links to the file.

Look at this output - Yours will look slightly different:

```
    drwxrwxr-x  3 joe  joe        4096 Oct 22 11:43 .
    drwxrwxr-x 10 joe  joe        4096 Oct 20 11:20 ..
    -rw-rw-r--  2 joe  joe          22 Oct 22 11:23 inode-hardink
    lrwxrwxrwx  1 joe  joe          14 Oct 22 11:21 inode-symlink -> sunshine.txt
    drwxrwxr-x  6 joe  joe        4096 Oct  6 10:15 it1100
    -rw-rw-r--  1 joe  joe          64 Oct 22 11:21 rain-hardlink
    lrwxrwxrwx  1 joe  joe           7 Oct 22 11:43 sun-symlink -> sun.txt
    -rw-rw-r--  1 joe  joe          64 Oct 22 11:43 wind.txt
```

Inode Metadata

- We should see that each directory has at least (2) hard links, one for the name reference contained in the parent and one for the reference to self `.`. It will also have one for every child directory that

references its parent with a `..`. And one for each hardlink to the directory.
- Each file will have at least (1) hard link - the filename itself that we see here in the parent directory.
- What does the 3 mean on line 1?
- How about the 10 and 6? What can we surmise about these directories?
- Where do these hardlinks most likely exist?
- Why does the file inode-hardlink have a 2 instead of just a 1?
- Why doesn’t rain-hardlink have a 2 also?
- Why doesn’t inode-symlink have a 2 also?