THE VIRTUAL FILE SYSTEM

- "Any problem in computer science can be solved with another level of indirection“

  -Butler Lampson

Why we love Linux

In a word: Versatility

- How is it that Linux can host such a large array of formats?
- Why does Linux adapt so quickly to different file types, regardless of origin or distribution?
- Answer: the VFS

So what is the VFS?

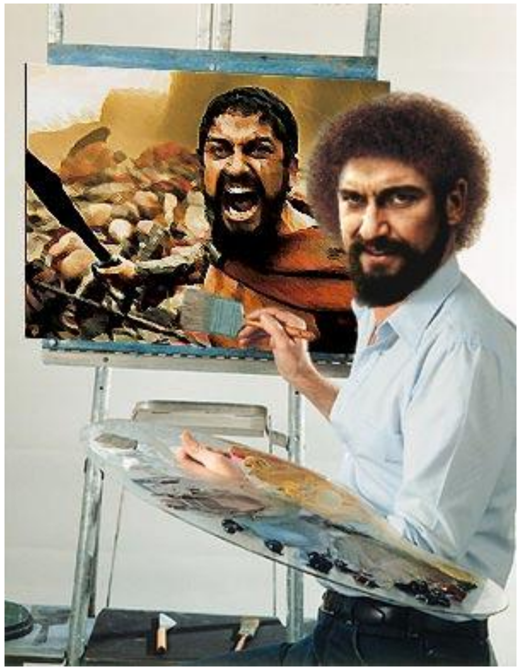
- The Virtual Filesystem (also known as the Virtual Filesystem Switch, or *VFS*) is a software layer that handles all syscalls pertaining to interaction with a standard Unix filesystem.

Meaning?

- Any time you work with a file, you do it through the VFS
- The twist: We define "file" very broadly and very abstractly

Why this is a big deal

- Unix-like operating systems (read: Linux) look at almost everything as if it were a file.
- To this end they have classified most entities into 5 standard file types:
  - Files
  - Directories
  - Pipes
  - Device files
  - Symbolic links



Uh huh. So?

- These file types are wildly different from each other.
- Implementation, functionality, purpose. All completely different.
- BUT…
- The VFS allows us to treat them all the same
- Needless to say, this is a time saver

What now

Now, a demonstration of my power

“stat” is a cute little command line tool that can show you file or filesystem status.

- The flags we shall use are:
  - ‘L’ - Dereference flag. Follow links
  - ‘f’ - Filesystem flag. Check filesystem status instead of file status
  - ‘c’ - Indicates that we are about to choose a specific format for displaying the data
  - ‘%T’ - Displays minor device type in hexadecimal

The classic example

- Let’s take a look at two directories mounted in very different places:
  - /media/KINGSTON (a USB flash drive)
  - /home/user/Desktop (the directory you keep all your passwords in)
- Write the following commands

```
$ stat -f -L -c %T /media/KINGSTON
$ stat -f -L -c %T /home/user/Desktop
```

- Calling these commands renders as output “msdos” and “ext2/ext3” respectively

What does this teach us

- Well, it tell us that KINGSTON is a mounted on an msdos file system
- It tells us that our Desktop is, in fact, an Extended Filesystem Directory
- Most importantly, this information should at least hint at the fact that we are dealing with two different, potentially incompatible, technologies

Now we try to mix it up

- You want to copy a directory from a USB flash drive to some directory on your computer

```
$ cp /media/KINGSTON/answers2OSFinal  /home/user/Desktop
```

- Incredibly, it works! This is because the ‘cp’ program is not required to know how to interact with the specific filesystem types of each directory. It need only interact with the VFS that encapsulates them and their functionality.
- Neato

So what file systems can VFS handle?

- Mainly three types and their multiple derivates and spinoffs
  - Network file systems
    - NFS
    - CIFS (windows)
  - Special file systems
    - /proc
    - /sys
  - Disc-Based file systems
    - Ex2
    - Ex3
    - Msdos

At the heart of the abstraction

- The main idea behind VFS is using the common file model which represents all supported filesystems
- This construct is built to be self-reinforcing so the assumptions the VFS makes about a given file system are cobbled together on-the-fly at run time

System calls vs the VFS

- Not really a 1 to 1 comparison, but there is one major similarity
- Both offer a direct line to the kernel
  - Syscalls do this with their syscall handler
  - VFS filesystems (like /proc and /sys) do this by hooking into kernel code with things like kobjects
- The net effect of this is having two potential API’s for kernel services