Virtual filesystems:
why we need them and how they work

Alison Chaiken
Peloton Technology
alison@she-devel.com

March 9, 2019

My coworkers with our product

We're hiring.

Agenda

- Filesystems and VFS
- /proc and /sys
- Monitoring with eBPF and bcc
- About bind mounts and namespaces
- containers and ro-rootfs
- live-media boots

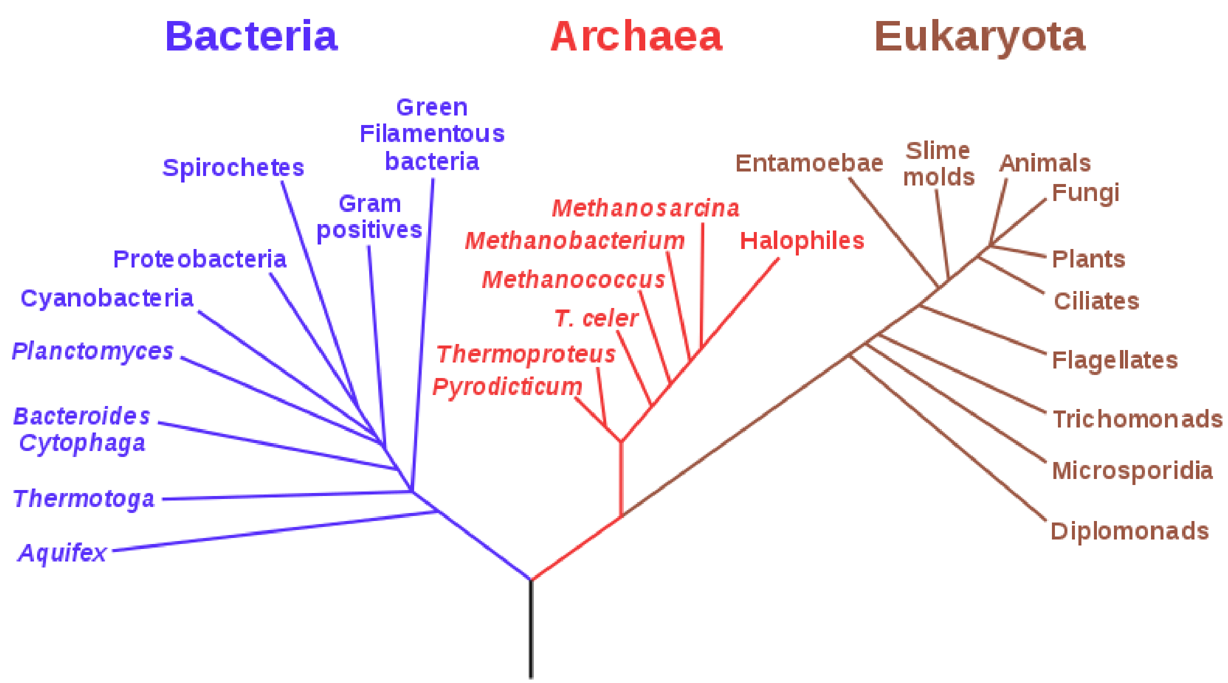
Does your system work now?

Do you really want to mess with it?

What is a filesystem?

What is a filesystem?

- Robert Love: “A *filesystem* is a hierarchical storage of data adhering to a specific structure.”



Does the image depict a filesystem?

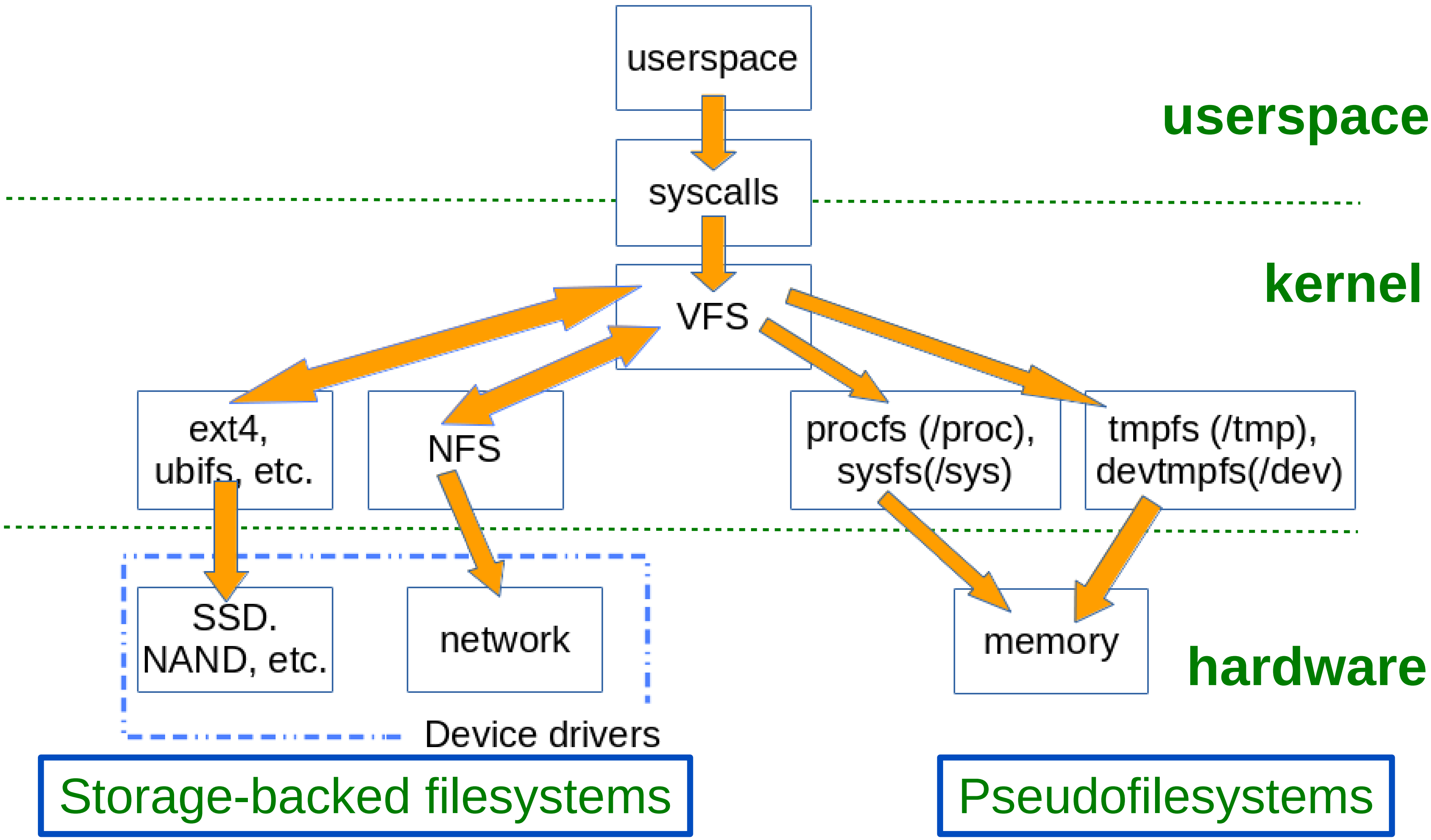
Linux's definition of a filesystem

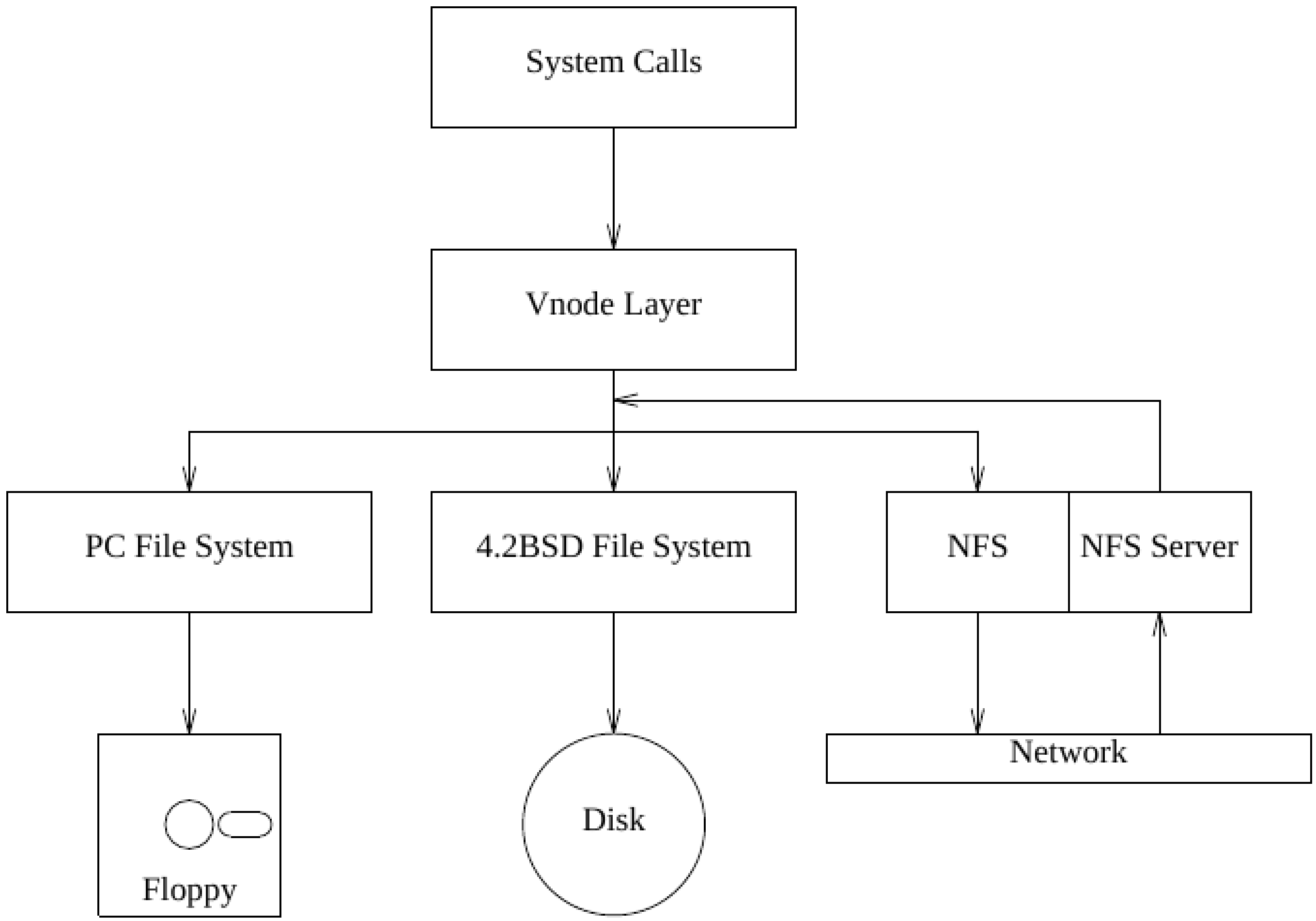
A filesystem *must* define the system calls:

```
struct file_operations {
    …
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    int (*open) (struct inode *, struct file *);
    …
}
```

What are virtual filesystems?

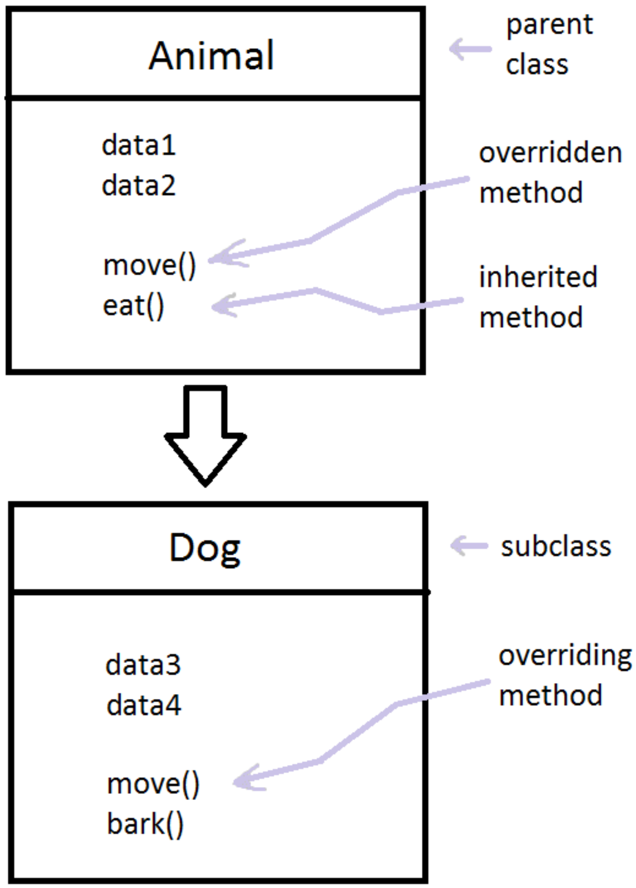
How VFS are used



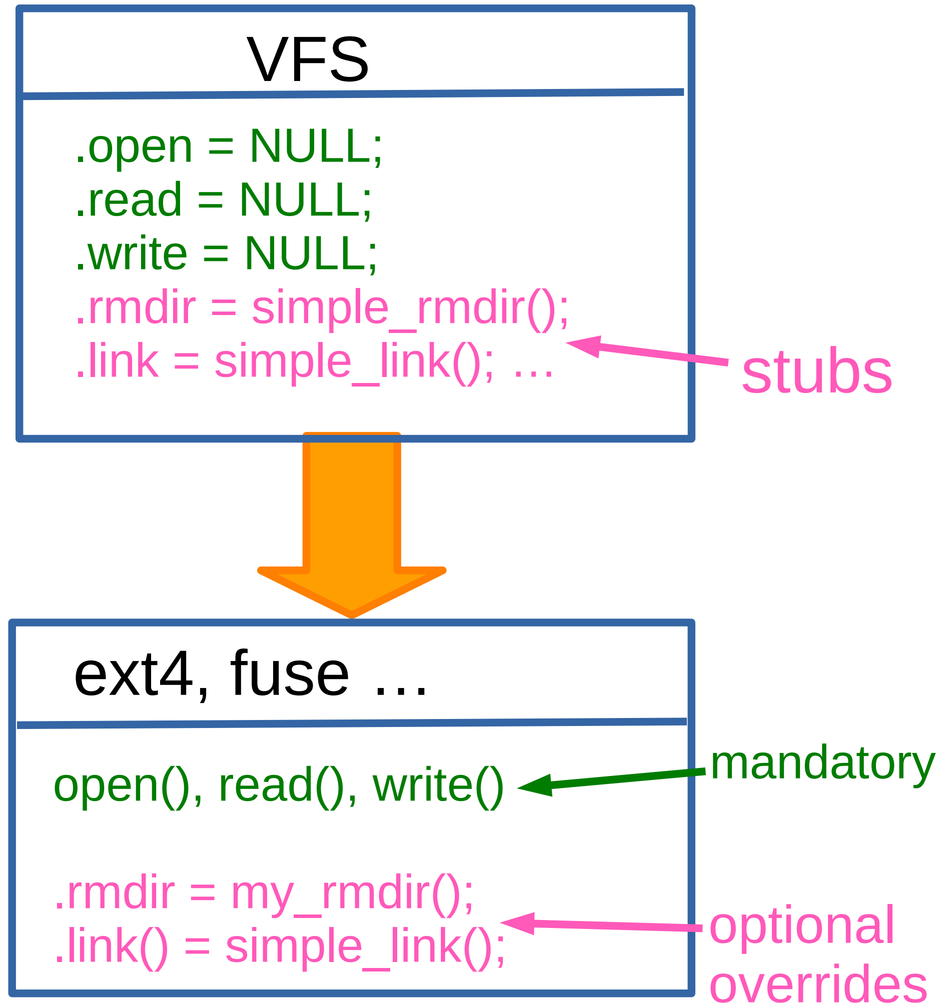


S. R. Kleiman and Sun Microsystems,
"Vnodes: An Architecture for Multiple File System Types",
in Proc. USENIX, Summer 1986.

VFS are an abstract interface that specific FS's implement



https://commons.wikimedia.org/w/index.php?curid=64193508



Typical file_operations

```
struct file_operations
ext4_file_operations = {
  .llseek       = ext4_llseek,
  .read_iter = ext4_file_read_iter,
  .write_iter = ext4_file_write_iter,
  .unlocked_ioctl = ext4_ioctl,
  .mmap = ext4_file_mmap,
  .mmap_supported_flags =
      MAP_SYNC,
  .open  = ext4_file_open,
  .release   = ext4_release_file,
  .fsync      = ext4_sync_file,
  .get_unmapped_area =
     thp_get_unmapped_area,
  .splice_read  =
     generic_file_splice_read,
  .splice_write  =
     iter_file_splice_write,
  .fallocate  = ext4_fallocate,
};
```

VFS Basics

- The VFS methods are defined in the kernel's fs/*c source files.
- Subdirectories of fs/ contain specific FS implementations.
- VFS resolve paths and permissions before calling into FS methods.
- A great example of code reuse! Unless …

Kernel quality control, or the lack thereof

By **Jonathan Corbet**
December 7, 2018

Filesystem developers tend toward a high level of conservatism when it comes to making changes; given the consequences of mistakes, this seems like

“Resources limits were not respected, users could overwrite a setuid file without resetting the setuid bits, time stamps would not be updated . . . affected all filesystems offering those features and needed to be fixed at the VFS level.”

Link to article

/proc and /sys

The observation that motivated the talk

Try this:

```
$ stat /proc/cpuinfo
$ stat /sys/power/state
$ file /proc/cpuinfo
$ file /sys/power/state
```

Why are the results so different?

```
$ stat /sys/bus/usb/uevent
  File: /sys/bus/usb/uevent
  Size: 4096            Blocks: 0          IO Block: 4096   regular file
Device: 13h/19d Inode: 9953        Links: 1
Access: (0200/--w-------)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2019-01-19 07:39:18.564000317 -0800
Modify: 2019-01-19 07:39:18.564000317 -0800
Change: 2019-01-19 07:39:18.564000317 -0800
 Birth: -
$
$ stat /proc/interrupts
  File: /proc/interrupts
  Size: 0               Blocks: 0          IO Block: 1024   regular empty file
Device: 4h/4d   Inode: 4026532036  Links: 1
Access: (0444/-r--r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2019-01-19 12:27:36.029101020 -0800
Modify: 2019-01-19 12:27:36.029101020 -0800
Change: 2019-01-19 12:27:36.029101020 -0800
 Birth: -
$
```

Sysfs

System boot

Procfs

Now

/procfs has tables; /sys has single params

```
$ head /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
  0:          8          0          0          0  IR-IO-APIC   2-edge      timer
  1:          0          0          0          9  IR-IO-APIC   1-edge      i8042
  8:          0          1          0          0  IR-IO-APIC   8-edge      rtc0
  9:          0      11603          0          0  IR-IO-APIC   9-fasteoi   acpi
 12:          0          0        602          0  IR-IO-APIC  12-edge      i8042
 18:          0          1          0          0  IR-IO-APIC  18-fasteoi   i801_smbus
 23:          0          0          0         35  IR-IO-APIC  23-fasteoi   ehci_hcd:usb3
 40:          0          0          0          0  DMAR-MSI   0-edge      dmar0
 41:          0          0          0          0  DMAR-MSI   1-edge      dmar1
$
$ cat /sys/kernel/boot_params/version
0x020d
$
```

state of kernel itself is visible via procfs

- /proc/<PID> directories contain per-process stats.
- The 'sysctl' interface manipulates /proc/sys:

    $ 'sysctl -a' lists system memory, network tunables

- procfs files are 'seq files' whose contents are generated dynamically.

/proc files: empty or no?

```
# head /proc/meminfo
MemTotal:        2061484 kB
MemFree:         1988796 kB
MemAvailable:    1996732 kB
Buffers:               0 kB
Cached:            36988 kB
SwapCached:            0 kB
Active:            23780 kB
Inactive:          22332 kB
Active(anon):       9320 kB
Inactive(anon):     8628 kB
#
# wc -l /proc/meminfo
40 /proc/meminfo
#
# ls -lh /proc/meminfo
-r--r--r-- 1 root root 0 Jan 15 19:59 /proc/meminfo
#
```

The contents of procfs appear when summoned

PHYSICS TODAY / APRIL 1985 PAG. 38-47

Is the moon there when nobody looks? Reality and the quantum theory

"It is a fundamental quantum doctrine that a measurement does not reveal a pre-existing value of the measured property." -- David Mermin

sysfs is how the kernel reacts to events

- sysfs:
  - publishes *events* to userspace about appearance and disappearance of devices, FS, power, modules …
  - allows these objects to be configured.
  - includes the kernel's famous stable ABI.

- In sysfs lies the userspace that one MUST NOT BREAK!

Watch USB stick insertion with eBPF and bcc

git clone git@github.com:iovisor/bcc.git

```
$ sudo ./trace.py 'p::sysfs_create_files'
PID     TID     COMM            FUNC
7       7       kworker/u16:0   sysfs_create_files
^C
```

trace.py source

Use tplist.py to discover kprobes and userspace probes that trace.py can watch.

Illustrating the full power of bcc-tools

```
$ sudo ./trace.py -K -I /usr/src/linux-source-4.19/include/linux/sysfs.h 'p::sys
fs_create_files(struct kobject *kobj, const struct attribute **ptr) "Created fil
ename is %s", (*ptr)->name'
PID     TID     COMM            FUNC             -
7711    7711    kworker/u16:3   sysfs_create_files Created filename is events
        sysfs_create_files+0x1 [kernel]
        __device_add_disk+0x2ee [kernel]
        sd_probe_async+0xf5 [kernel]
        async_run_entry_fn+0x39 [kernel]
        process_one_work+0x1a7 [kernel]
        worker_thread+0x30 [kernel]
        kthread+0x112 [kernel]
        ret_from_fork+0x35 [kernel]

^C
```

Watch the same sysfs_create_files() function, get more details.

The source code tells you what programs *can* do; eBPF/bcc-tools tell you what they *actually* do.

	Kernel	User space	easy to use	Minimal performance hit
ftrace	X			?
strace		X	X	
bcc/ eBPF	X	X	X	X

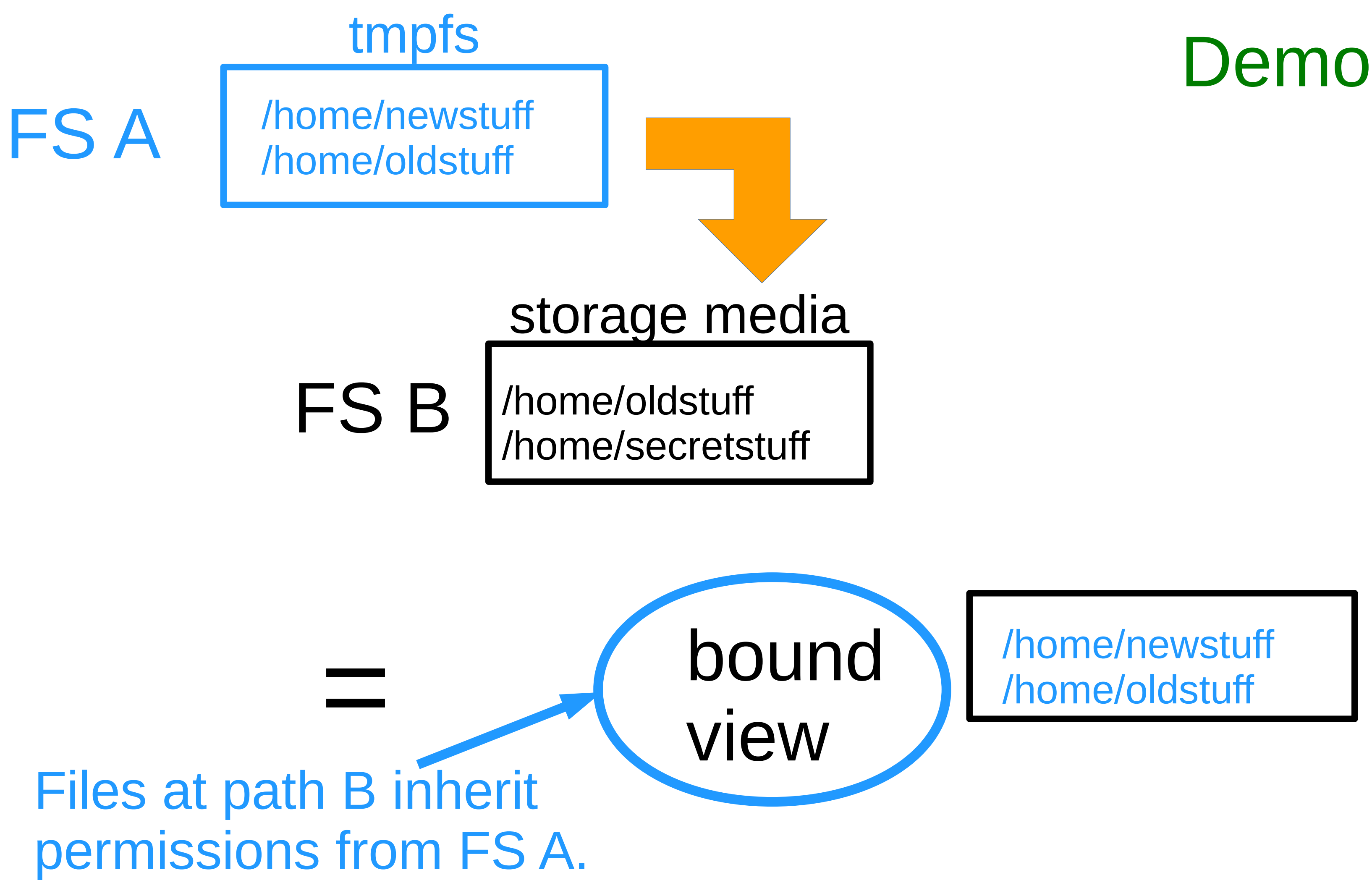
Bind mounts
and
mount namespaces

symlinks, chroots, binds and overlays

- Symlinking a file or directory provides no security, and is static.
- chroot / is dynamic, but provides no /proc, /sys, /dev.
- Bind-mounting a *file* or *directory* over another:
  - provides dynamic, secure, granular *reference* to dir/file at another path;
  - useful for containers and IoT devices.
- Overlaying a *filesystem* over another:
  - provides a *union* of the FS at one path with the FS at another;
  - useful for live media boots.

Bind mount

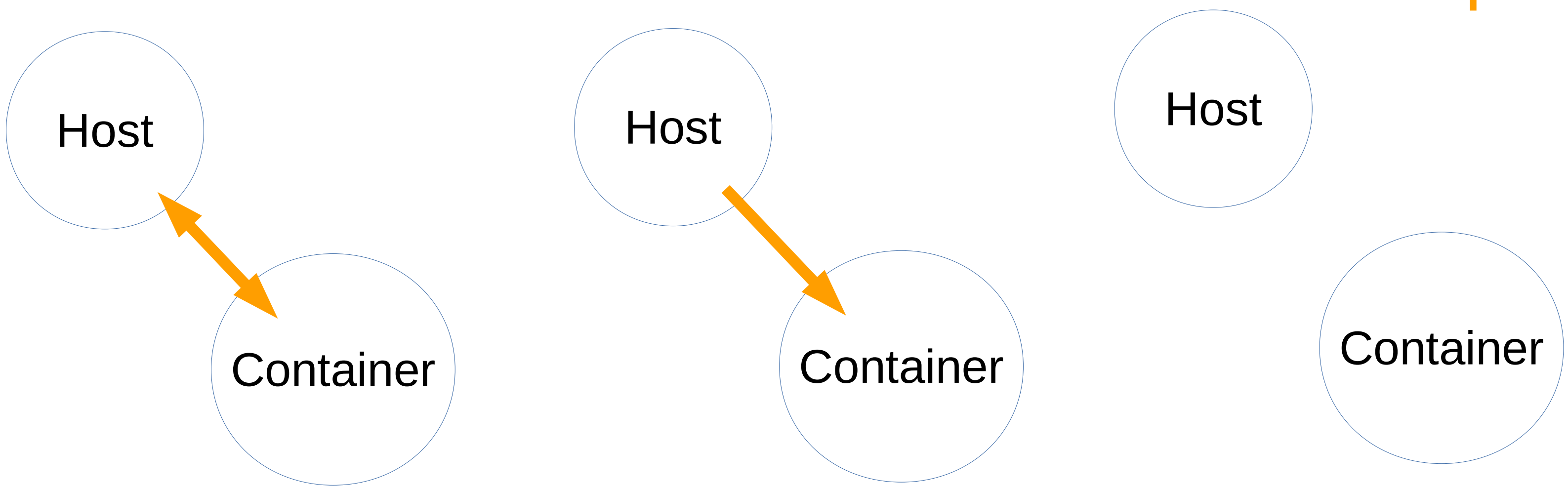
Demo



Bind-mount flags control visibility of mount events, not files

```
A given mount can be in one of the following states
1) shared
2) slave
3) shared and slave
4) private
5) unbindable
```

prevent loops



Shared (mirror)

Slave (my mounts are private)

Private (default)

From Documentation/filesystems/sharedsubtree.txt.

Namespaces are magic that enables containers

- chroot, the old 'container', had minimal security.
- Container security is implemented (in part) via *namespaces*.
- Each container can have a different view of the system's files.
- See an overview with mountinfo files.
- Info about fields is in Documentation/filesystems/proc.txt.

Example: containers

Start a simple container

```
$ sudo systemd-nspawn -D /srv/nspawn/
Spawning container nspawn on /srv/nspawn.
Press ^] three times within 1s to kill container.
root@nspawn:~#
root@nspawn:~# wc -l /proc/kallsyms
0 /proc/kallsyms
root@nspawn:~# head /proc/kallsyms
root@nspawn:~#
```

systemd-nspawn is a container manager akin to runc or lxc.

Watch container bind mounts with BCC

```
$ sudo ./mountsnoop.py
COMM                 PID      TID      MNT_NS       CALL
```

```
systemd-nspawn   14911   14911   4026532592  mount("/srv/nspawn", "/", "", MS_NOSUID|MS_NOEXEC|MS_REMOUNT|MS_BIND|MS_REC|MS_POSIXACL|MS_PRIVATE|MS_KERNMOUNT|MS_STRICTATIME|0x7f301c000300, "") = 0
systemd-nspawn   14912   14912   4026532593  mount("proc", "/proc", "proc", MS_NOSUID|MS_NOEXEC|MS_REMOUNT|MS_BIND|MS_REC|MS_POSIXACL|MS_PRIVATE|MS_KERNMOUNT|MS_STRICTATIME|0x7f301c000300, "") = 0
systemd-nspawn   14912   14912   4026532593  mount("/proc/sys", "/proc/sys", "", MS_NOSUID|MS_NOEXEC|MS_REMOUNT|MS_BIND|MS_REC|MS_POSIXACL|MS_PRIVATE|MS_KERNMOUNT|MS_STRICTATIME|0x7f301c000300, "") = 0
systemd-nspawn   14912   14912   4026532593  mount("", "/proc/sys", "", MS_NOSUID|MS_NOEXEC|MS_REMOUNT|MS_BIND|MS_REC|MS_POSIXACL|MS_PRIVATE|MS_KERNMOUNT|MS_STRICTATIME|0x7f301c000300, "") = 0
systemd-nspawn   14912   14912   4026532593  mount("/tmp/.#inaccessiblea5dc6c3941d65f6d", "/proc/kallsyms", "", MS_NOSUID|MS_NOEXEC|MS_REMOUNT|MS_BIND|MS_REC|MS_POSIXACL|MS_PRIVATE|MS_KERNMOUNT|MS_STRICTATIME|0x7f301c000300, "") = 0
```

Intentional hiding of kernel symbols

Private mounts: invisible to parent

read-only root filesystems

Read-only rootfs: a critical tool for embedded

https://tinyurl.com/y7t2k7ma

Motivation:

- Safely yank device power.
- rootfs does not get full.
- Malware cannot modify /usr/, /etc, keys . . .
- Device problems reported from the field reproduce.
- Forces separation of application data and binaries.

read-only rootfs challenges

- /var must be mounted separately from /.
- Programs that modify $HOME at runtime: gstreamer, openssh-client …
- rootfs builders must
  - pre-populate these files, or
  - bind- or overlay-mount them from other paths.

Not a bug but a feature!

Overlayfs

tmpfs

upper

/etc/passwd
/home/newstuff

storage media

lower

/etc/passwd
/home/oldstuff

=

overlaid view

/etc/passwd
/home/oldstuff
/home/newstuff

Replace /etc/passwd inside a container

```
$ mkdir /tmp/upperdir
$ mkdir /tmp/workdir
$ cp passwd /tmp/upperdir/
$ sudo mount -t overlay overlay -oupperdir=/tmp/upperdir/,workdir=/tmp/workdir/,lowerdir=/etc /etc
$ ls /etc | head
abcde.conf
acpi/
adduser.conf
adjtime
aliases
aliases.db
alsa/
alternatives/
anacrontab
apache2/
$ whoami
dennis
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
dennis:x:1000:1000:Dennis Ritchie,,,:/home/dennis:/bin/bash
$
```

Summary

- VFS are one of Linux' core components.
- /proc, /sys and most on-HW FS are based on VFS.
- Bind-mounts and mount NS enable containers and ro-rootfs.
- bcc-tools and eBPF are remarkably powerful and easy to use.

Acknowledgements

Much thanks to

Akkana Peck, Michael Eager

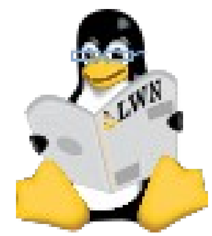
and Sarah Newman for comments and corrections.

Ballroom H at 6 PM:

"Accidentally accessible"

References

- About kobjects, seq files and sysfs: Appendix C, Essential Device Drivers by S. Venkateswaran



- About “everything is a file”: chapters 2, 4, 13, Linux Kernel Development by Robert Love
- Excellent mount namespaces article by Michael Kerrisk
- Excellent “Object-oriented design patterns in the kernel” article series by Neil Brown
- “BPF in the Kernel” series by Matt Fleming

Example: Live CD

Prepopulated /run directory on Kali Linux LiveCD

```
$ sudo mount -o ro,loop kali-linux-2019-W09-amd64.iso /mnt/iso
$ ls /mnt/iso
autorun.inf  dists/   firmware/  install/   md5sum.txt  tools/
boot/        EFI/     g2ldr      isolinux/  pool/       win32-loader.ini
debian@      efi.img  g2ldr.mbr  live/      setup.exe
$ ls /mnt/iso/live
filesystem.packages         initrd.img-4.19.0-kali1-amd64
filesystem.packages-remove  memtest
filesystem.size             vmlinuz
filesystem.squashfs         vmlinuz-4.19.0-kali1-amd64
initrd.img
$ sudo mount -o ro,loop /mnt/iso/live/filesystem.squashfs /mnt/squashfs/
$ ls /mnt/squashfs/
0      etc/             lib@     media/  root/  sys/  vmlinuz@
bin@   home/            lib32@   mnt/    run/   tmp/  vmlinuz.old@
boot/  initrd.img@      lib64@   opt/    sbin@  usr/
dev/   initrd.img.old@  libx32@  proc/   srv/   var/
$ ls /mnt/squashfs/run
apache2/  exim4/   lock/  mount/       samba/   speech-dispatcher/  utmp
dnsmasq/  iodine/  lvm/   postgresql/  screen/  stunnel4/
$ ls /mnt/squashfs/run/samba/
msg.lock/  names.tdb  upgrades/
$
```

Kali Linux relies on overlayfs

```
S sudo mount -o ro,loop kali-linux-2019-W09-amd64.iso /mnt/iso
S sudo mount -o ro,loop /mnt/iso/live/filesystem.squashfs /mnt/squashfs
S ls /mnt/squashfs/usr/lib/live/boot
0001-init-vars.sh*          9990-mount-cifs.sh*
0010-debug*                 9990-mount-http.sh*
0020-read-only*             9990-mount-iscsi.sh*
0030-verify-checksums*      9990-mount-nfs.sh*
2010-remove-persistence*    9990-netbase.sh*
3020-swap*                  9990-netboot.sh*
9990-cmdline-old*           9990-networking.sh*
9990-fstab.sh*              9990-overlay.sh*
9990-initramfs-tools.sh*    9990-select-eth-device.sh*
9990-main.sh*               9990-toram-todisk.sh*
9990-misc-helpers.sh*
S head /mnt/squashfs/usr/lib/live/boot/9990-overlay.sh
#!/bin/sh

#set -e

setup_unionfs ()
{
        image_directory="${1}"
        rootmnt="${2}"
        addimage_directory="${3}"

S
```

Info from /proc/<PID>/mountinfo about shared mounts

```
root@nspawn:~# cat /proc/1/mountinfo
1041 950 8:1 /srv/nspawn / rw,relatime shared:482 master:1 - ext4 /dev/sda1 rw,errors=remount-ro
1042 1041 0:52 / /tmp rw,nosuid,nodev shared:483 - tmpfs tmpfs rw
1043 1041 0:18 / /sys ro,nosuid,nodev,noexec,relatime shared:484 - sysfs sysfs rw
1044 1041 0:67 / /dev rw,nosuid shared:485 - tmpfs tmpfs rw,mode=755
1045 1044 0:69 / /dev/shm rw,nosuid,nodev shared:486 - tmpfs tmpfs rw
1046 1044 0:17 / /dev/mqueue rw,relatime shared:488 - mqueue mqueue rw
1047 1044 0:71 / /dev/pts rw,nosuid,noexec,relatime shared:489 - devpts devpts rw,gid=5,mode=620,ptmxmode=666
1048 1044 0:19 /4 /dev/console rw,nosuid,noexec,relatime shared:490 master:3 - devpts devpts rw,gid=5,mode=620,ptmxm
ode=000
1049 1041 0:70 / /run rw,nosuid,nodev shared:487 - tmpfs tmpfs rw,mode=755
1050 1049 0:20 /systemd/nspawn/propagate/nspawn /run/systemd/nspawn/incoming ro,relatime master:5 - tmpfs tmpfs rw,s
ize=785436k,mode=755
1053 1041 0:73 / /proc rw,nosuid,nodev,noexec,relatime shared:491 - proc proc rw
1054 1053 0:73 /sys /proc/sys ro,nosuid,nodev,noexec,relatime shared:491 - proc proc rw
1055 1053 0:52 /.#inaccessible9dc31fd0ad5399ef//deleted /proc/kallsyms ro,nosuid,nodev,noexec shared:483 - tmpfs tmp
fs rw
```

sysfs vs procfs sizes

```
$ find /proc -type f -size +1c 2>/dev/null
/proc/config.gz
$
$ find /sys -type f -size +1c 2>/dev/null | wc -l
12736
```

/sys files are 1 page of memory and contain 1 string/number.

/procfs files often 'contain' a table of data.

Overlayfs mounts

- Overlay mounts are like bind mounts, but changes in the upper directory obscure those in the lower directory.
- A file in /tmp/upper can appear to replace files in /home on storage media.

Bind mounts

- Bind mounts make an existing file or directory appear at a new path.
  - Changes to the directory appear in both places.
  - A file in /tmp can appear to be in $HOME in addition to files that are in $HOME on storage media.

Subtle but important win with ro-rootfs

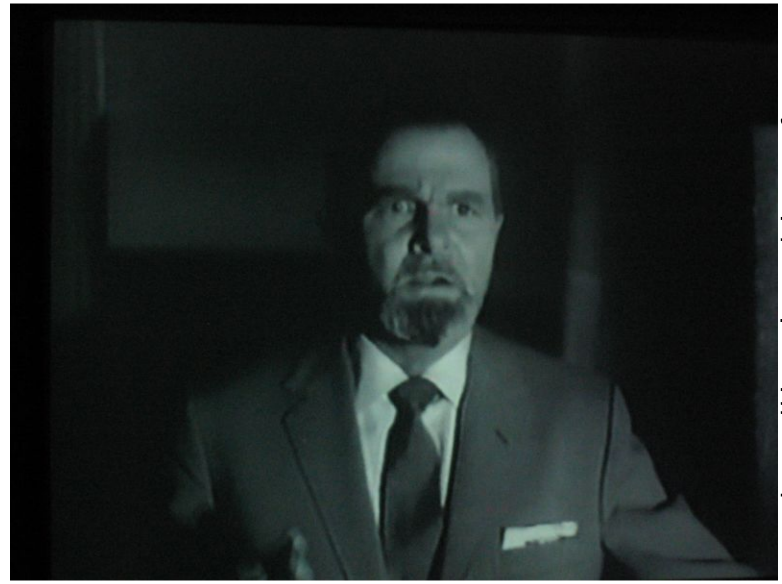
A ro-rootfs forces better application design via separation of data and binaries.

A systems administration tip!

- Try this:

  $ findmnt /tmp

- Is /tmp on /dev/sdx? on /dev/hdx?
- Fix by editing /etc/fstab!



https://tinyurl.com/ybomxyfo

```
$ grep tmpfs /etc/fstab
tmpfs                                   /tmp            tmpfs   defaults        0
 0
```

Keep a copy of /etc/fstab on a bootable USB stick.
Make sure that fstab ends with a newline!

Turning off sysfs?

```
Designers of embedded systems may wish to say N here to conserve space.

Symbol: SYSFS [=y]
Type  : boolean
Prompt: sysfs file system support
  Location:
    -> File systems
      -> Pseudo filesystems
  Defined at fs/sysfs/Kconfig:1
  Selects: KERNFS [=y]
  Selected by: AT91_ADC [=n] && IIO [=y] && ARCH_AT91 [=n] && INPUT [=y] || CONFIGFS_FS [=y
```

Keyboard and mouse

A few oddities: /proc/kcore

```
$ sudo gdb -q vmlinux /proc/kcore
Reading symbols from vmlinux...done.
[New process 1]
Core was generated by `BOOT_IMAGE=/boot/vmlinuz-4.13.13 root=UUID=c7d53478-7054-470b-9
f37-bbb20a5e7036'.
#0  0x0000000000000000 in irq_stack_union ()
(gdb) bt
#0  0x0000000000000000 in irq_stack_union ()
#1  0x0000000000000000 in ?? ()
(gdb) l
1       /*
2        *  linux/arch/x86/kernel/head_64.S -- start in 32bit and switch to 64bit
3        *
4        *  Copyright (C) 2000 Andrea Arcangeli <andrea@suse.de> SuSE
5        *  Copyright (C) 2000 Pavel Machek <pavel@suse.cz>
6        *  Copyright (C) 2000 Karsten Keil <kkeil@suse.de>
7        *  Copyright (C) 2001,2002 Andi Kleen <ak@suse.de>
8        *  Copyright (C) 2005 Eric Biederman <ebiederm@xmission.com>
9        */
10
(gdb) 
```