CS615 - System Administration

Filesystem and OS Boot Process Concepts

Department of Computer Science
Stevens Institute of Technology
Jan Schaumann
jschauma@stevens.edu
https://stevens.netmeister.org/615/

Basic Filesystem Concepts

The UNIX Filesystem

Basic Filesystem Concepts: The UNIX Filesystem

The filesystem is responsible for storing the data on the disk. So to read/write data, it needs to know in which physical blocks the actual data is located; ie how to map files to the disk blocks.

Let's pretend we're a filesystem...

```
aws ec2 create-volume --size 1 --availability-zone us-east-1d

aws ec2 attach-volume --volume-id  XXX --instance-id XXX --device /dev/sda3

dmesg

for i in $(seq $((1024 * 128))); do
        printf '\0\0\0\0\0\0\0\0'
done | dd of=/dev/xbd2d
```

Basic Filesystem Concepts: The UNIX Filesystem

Basic Filesystem Concepts: The UNIX Filesystem

Basic Filesystem Concepts: The UNIX Filesystem

Basic Filesystem Concepts: The UNIX Filesystem

Let's pretend we're a filesystem...

Creating a 'file':

```
echo -n meow | dd of=/dev/xbd2d

dd if=/dev/xbd2d count=512 2>/dev/null | hexdump -C
```

Basic Filesystem Concepts: The UNIX Filesystem

Let's pretend we're a filesystem...

Each file can be up to 4096 bytes in size:

```
echo -n meow | dd of=/dev/xbd2d

dd if=/dev/xbd2d 2>/dev/null | hexdump -C

echo -n miumiu | dd of=/dev/xbd2d bs=1 seek=4096

dd if=/dev/xbd2d count=512 2>/dev/null | hexdump -C
```

Basic Filesystem Concepts: The UNIX Filesystem

Let's pretend we're a filesystem...

We can store data for multiple files in a single bucket:

```
printf 'meow' | dd of=/dev/xbd2d
printf 'miumiu' | dd of=/dev/xbd2d bs=1 seek=32

dd if=/dev/xbd2d count=512 2>/dev/null | hexdump -C
```

Basic Filesystem Concepts: The UNIX Filesystem

Let's pretend we're a filesystem...

We can store data for multiple files in a single bucket:

```
printf 'meow miu meow miu meow miu meow ' | dd of=/dev/xbd2d
printf 'miumiu miau miao miauw miaow ' | dd of=/dev/xbd2d bs=1 seek=32
printf 'hiss hiss' | dd of=/dev/xbd2d bs=1 seek=64
printf 'hiss hiss' | dd of=/dev/xbd2d bs=1 seek=96

dd if=/dev/xbd2d count=512 2>/dev/null | hexdump -C
```

Let's pretend we're a filesystem...

New format: identifier followed by data.

```
printf '\x1meow miu meow miu meow miu meow ' | dd of=/dev/xbd2d
printf '\x2miumiu miau miao miauw miaow ' | dd of=/dev/xbd2d bs=1 seek=512
printf '\x3hiss hiss' | dd of=/dev/xbd2d bs=1 seek=1024
printf '\x4hiss hiss' | dd of=/dev/xbd2d bs=1 seek=1535

dd if=/dev/xbd2d count=512 2>/dev/null | hexdump -C
```

Basic Filesystem Concepts: The UNIX Filesystem

Let's pretend we're a filesystem...

Quick note: removing a file does not "delete" the data:

```
dd if=/dev/rxbd2a 2>/dev/null | hexdump -C

rm /mnt/newfile

dd if=/dev/rxbd2a 2>/dev/null | hexdump -C
```

Basic Filesystem Concepts: The UNIX Filesystem

What's in a file?

Let's pretend we're a filesystem...

New format: identifier, permission, owner, group, data.

```
printf '\x1\x7\x4\x4\x0\x0meow miu meow miu meow miu meow ' | dd of=/dev/xbd2d
printf '\x2\x7\x4\x4\x1\x1purrrrrr' | dd of=/dev/xbd2d bs=1 seek=4096
```

Let's pretend we're a filesystem...

New format: identifier, permission, owner, group, data.

```
printf '\x1\x7\x4\x4\x0\x0meow miu meow miu meow miu meow ' | dd of=/dev/xbd2d
printf '\x2\x7\x4\x4\x1\x1purrrrrr' | dd of=/dev/xbd2d bs=1 seek=4096
```

Problem: what if we have a file >4K?

Let's pretend we're a filesystem...

New format: separate meta-data from data

- meta-data (16 bytes total, LE):
  - identifier (2 bytes)
  - permission (4 bytes)
  - owner (1 byte)
  - group (1 byte)
  - size (4 bytes)
  - offset (4 bytes)
- data stored in separate area

```
printf '\x1\x0\x0\x7\x4\x4\x0\x0\x20\x0\x0\x0\x0\x1\x0\x0' | dd of=/dev/xbd2d
printf 'meow miu meow miu meow miu meow ' | dd of=/dev/xbd2d bs=1 seek=4096
printf '\x2\x0\x0\x7\x4\x4\x1\x1\x8\x0\x0\x0\x20\x10\x0\x0' | dd of=/dev/xbd2d bs=1 s
printf 'purrrrrr' | dd of=/dev/xbd2d bs=1 seek=4128
```

(What limitations do we have now?)

A different ”filesystem” format

```
echo "hello world" > /root/newfile

tar cf - /root | dd of=/dev/xbd2d

dd if=/dev/xbd2d count=512 2>/dev/null | hexdump -C

dd if=/dev/xbd2d | tar tvf -
```

See also: `https://stevens.netmeister.org/615/tar.html`

Basic Filesystem Concepts: The UNIX Filesystem

The filesystem is responsible for storing the data on the disk. So to read/write data, it needs to know in which physical blocks the actual data is located; ie how to map files to the disk blocks.

Components of the Berkeley Fast Filesystem:

- set of *inode* storage cells

```
df -i
```

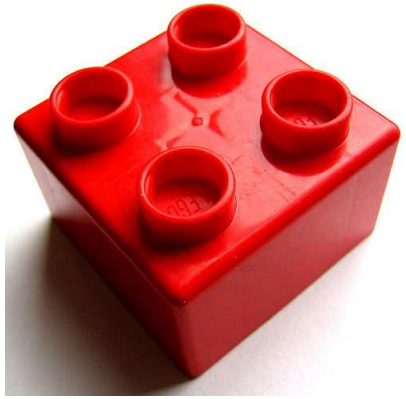
Basic Filesystem Concepts: The UNIX Filesystem

The filesystem is responsible for storing the data on the disk. So to read/write data, it needs to know in which physical blocks the actual data is located; ie how to map files to the disk blocks.

Components of the Berkeley Fast Filesystem:

- set of *inode* storage cells
- set of scattered “superblocks”

```
newfs -b 4096 -f 512 xbd2d
```



Basic Filesystem Concepts: The UNIX Filesystem

The filesystem is responsible for storing the data on the disk. So to read/write data, it needs to know in which physical blocks the actual data is located; ie how to map files to the disk blocks.

Components of the Berkeley Fast Filesystem:

- set of *inode* storage cells
- set of scattered “superblocks”
- map of disk blocks

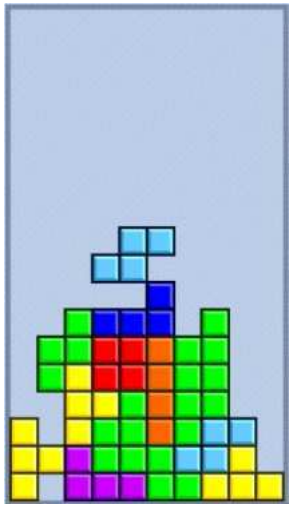
```
dumpfs -v xbd2
```

Basic Filesystem Concepts: The UNIX Filesystem

The filesystem is responsible for storing the data on the disk. So to read/write data, it needs to know in which physical blocks the actual data is located; ie how to map files to the disk blocks.

Components of the Berkeley Fast Filesystem:

- set of *inode* storage cells
- set of scattered “superblocks”
- map of disk blocks
- block usage summary



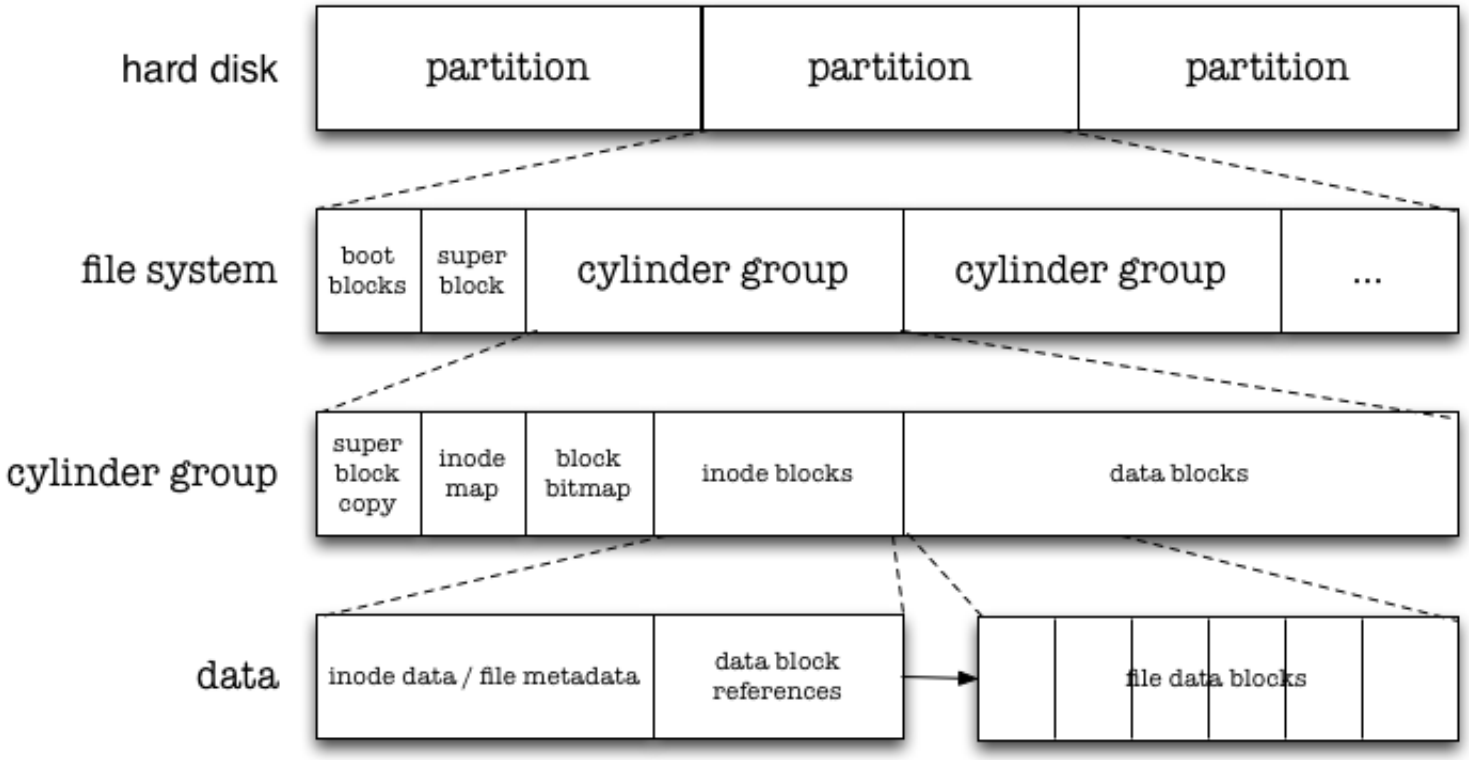
Basic Filesystem Concepts: The UNIX Filesystem

The filesystem is responsible for storing the data on the disk. So to read/write data, it needs to know in which physical blocks the actual data is located; ie how to map files to the disk blocks.

Components of the Berkeley Fast Filesystem:

- set of *inode* storage cells
- set of scattered “superblocks”
- map of disk blocks
- block usage summary
- set of data blocks

Basic Filesystem Concepts: The UNIX Filesystem



Basic Filesystem Concepts: The UNIX Filesystem

Information stored in an *inode*:

- user owner and group owner ID's
- file type
  - (regular ('r'), directory ('d'), special ('c', 'b'), symlink ('l'), socket ('s'), fifo ('p'))
- access mode (permissions)
- file access and modification time
- file status modification time
- number of links to the file
- size of the file
- disk device containing this file

```
$ stat /etc/passwd
```

Let's *not* pretend we're a filesystem...

```
dumpfs xbd2

disklabel xbd2

newfs -b 4096 -f 512 xbd2a

mount /dev/xbd2a /mnt

echo hello world > /mnt/newfile

dd if=/dev/rxbd2a 2>/dev/null | hexdump -C

dumpfs xbd2

dumpfs -v -i xbd2
```

Hooray!

5 Minute Break

Down the stack we go

Consider a website on an AWS EC2 instance...

Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server

Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
- require PHP, Perl, Ruby, Javascript, ...

Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
- require PHP, Perl, Ruby, Javascript, ...
- which uses generic library functions

Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
- require PHP, Perl, Ruby, Javascript, ...
- which uses generic library functions
- which make various system calls

Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
- require PHP, Perl, Ruby, Javascript, ...
- which uses generic library functions
- which make various system calls
- which the kernel handles for the OS

Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
- require PHP, Perl, Ruby, Javascript, ...
- which uses generic library functions
- which make various system calls
- which the kernel handles for the OS
- which is running in a virtual machine

Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
- require PHP, Perl, Ruby, Javascript, ...
- which uses generic library functions
- which make various system calls
- which the kernel handles for the OS
- which is running in a virtual machine
- which is running on top of a hypervisor

Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
- require PHP, Perl, Ruby, Javascript, ...
- which uses generic library functions
- which make various system calls
- which the kernel handles for the OS
- which is running in a virtual machine
- which is running on top of a hypervisor
- which uses firmware to manage various components

Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
- require PHP, Perl, Ruby, Javascript, ...
- which uses generic library functions
- which make various system calls
- which the kernel handles for the OS
- which is running in a virtual machine
- which is running on top of a hypervisor
- which uses firmware to manage various components
- which is running on some hardware

...and back up again

Bringin up this web service might include...

- power on hardware

...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization

...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader

...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader
- second stage boot loader

...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader
- second stage boot loader
- hypervisor kernel dom0 starts

...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader
- second stage boot loader
- hypervisor kernel dom0 starts
- domU is started

...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader
- second stage boot loader
- hypervisor kernel dom0 starts
- domU is started
- guest OS kernel starts

...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader
- second stage boot loader
- hypervisor kernel dom0 starts
- domU is started
- guest OS kernel starts
- kernel initializes (virtual) hardware

...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader
- second stage boot loader
- hypervisor kernel dom0 starts
- domU is started
- guest OS kernel starts
- kernel initializes (virtual) hardware
- `init(8)` (or similar) starts

...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader
- second stage boot loader
- hypervisor kernel dom0 starts
- domU is started
- guest OS kernel starts
- kernel initializes (virtual) hardware
- `init(8)` (or similar) starts
- system processes / daemons start

...and back up again

Bringin up this web service might include...

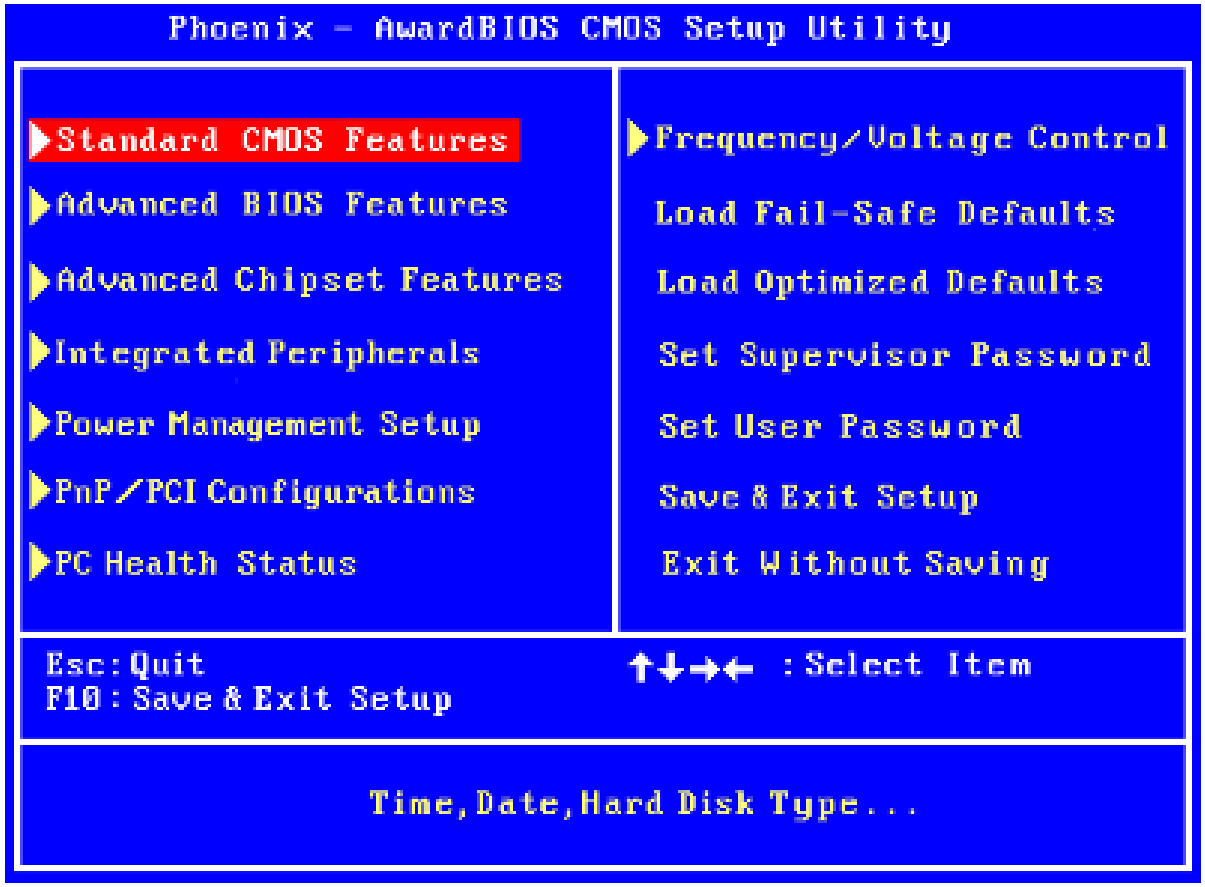
- power on hardware
- POST and other firmware initialization
- first stage boot loader
- second stage boot loader
- hypervisor kernel dom0 starts
- domU is started
- guest OS kernel starts
- kernel initializes (virtual) hardware
- `init(8)` (or similar) starts
- system processes / daemons start
- web server runs, binds network socket, serves content

Team Missions

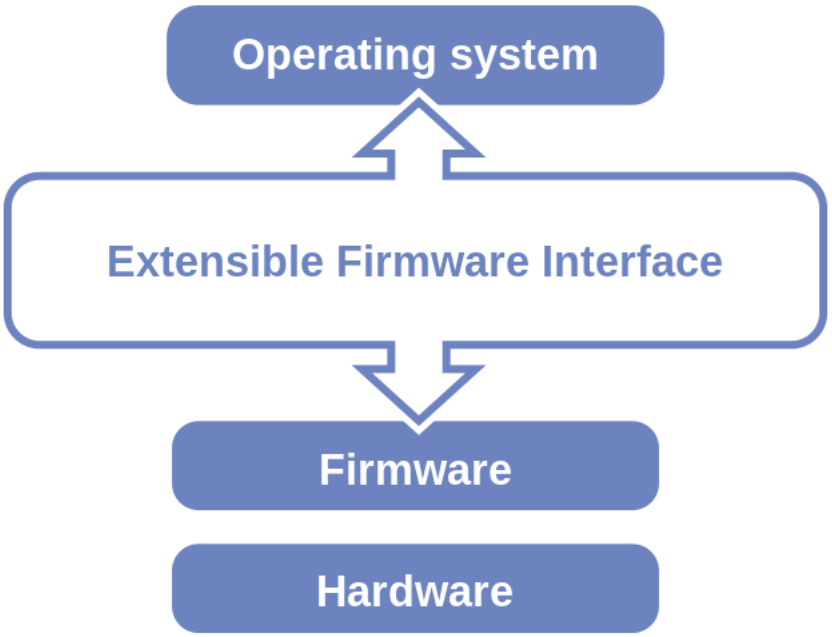
`https://is.gd/VFWyp6`

Typical Boot Sequence

Typical Boot Sequence



Typical Boot Sequence

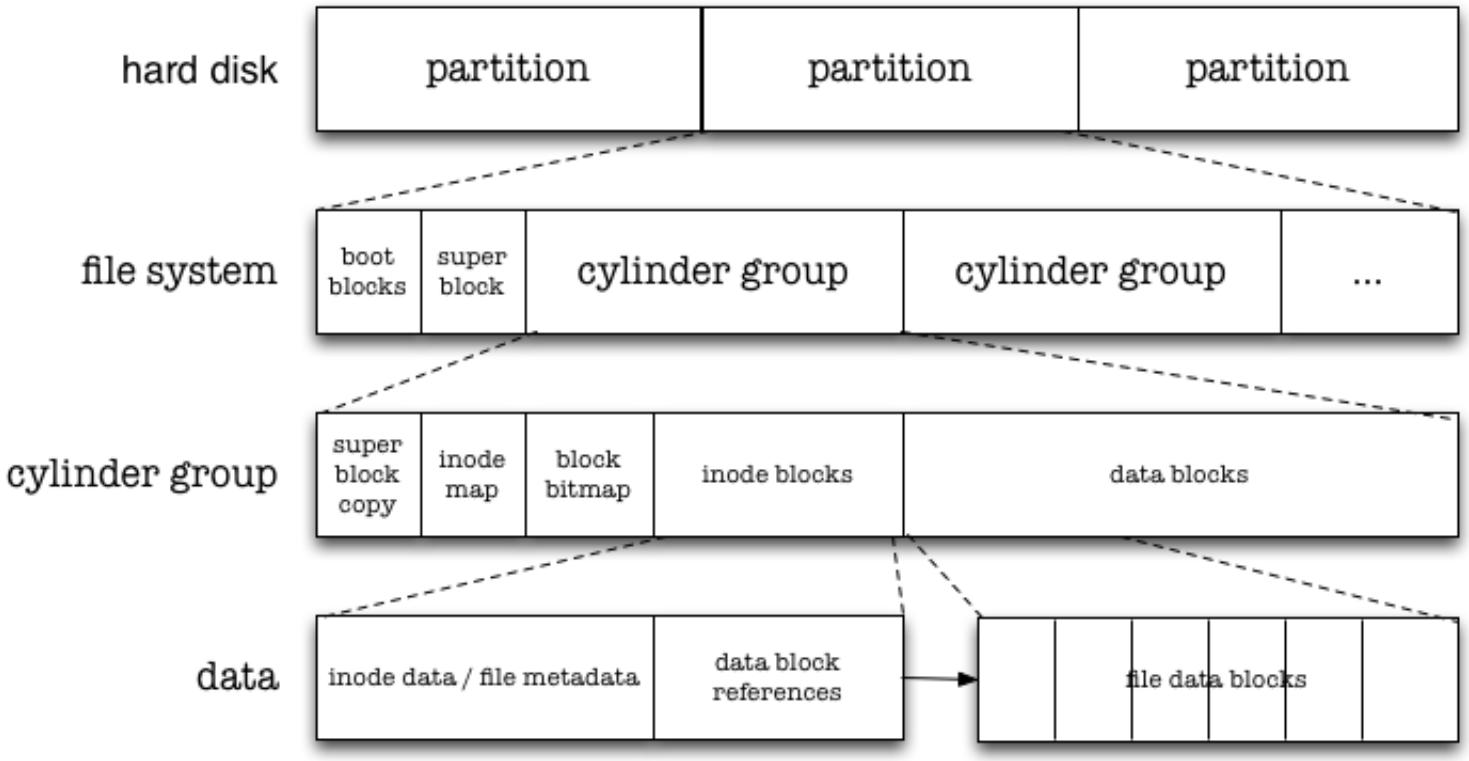


Unified Extensible Firmware Interface (UEFI)

Team Missions

https://is.gd/KgvP8p

Typical Boot Sequence



Typical Boot Sequence: BIOS and MBR

- first sector (512 bytes) of data storage device
- last two bytes contain signature `0x55 0xAA`
- 64 bytes allocated for partition table (four possible partitions at 16 bytes each)
- 446 bytes for primary boot loader code

Recall HW1

```
# fdisk /dev/xbd0
fdisk: primary partition table invalid, no magic in sector 0
Disk: /dev/xbd0d

BIOS disk geometry:
cylinders: 1023, heads: 255, sectors/track: 63 (16065 sectors/cylinder)
total sectors: 69206016

Partition table:
0: <UNUSED>
1: <UNUSED>
2: <UNUSED>
3: <UNUSED>
Bootselector disabled.
No active partition.
```

Boot partitions

See e.g.
https://en.wikipedia.org/wiki/Master_boot_record#Partition_table_entries

```
fdisk xbd2

# 0x80 = active; 0x00 inactive
printf '\x0' | dd of=/dev/xbd2d bs=1 seek=446
# CHS of first sector
printf '\01\01\00' | dd of=/dev/xbd2d bs=1 seek=447
# partition type 169 decimal = NetBSD
printf '\xa9' | dd of=/dev/xbd2d bs=1 seek=450
# CHS of last sector
printf '\x8a\x08\x82' | dd of=/dev/xbd2d bs=1 seek=451
# LBA of first sector (4 bytes, little endian)
printf '\x3f' | dd of=/dev/xbd2d bs=1 seek=454
# total # of sector (4 bytes little endian)
printf '\xc1\xff\x1f' | dd of=/dev/xbd2d bs=1 seek=458
# magic header / boot signature
printf '\x55\xaa' | dd of=/dev/xbd2d seek=510 bs=1

fdisk xbd2

# of, a bit easier:
fdisk -f -u -0 -s 169/63/2097089 /dev/rxbd2d
```

Boot partitions

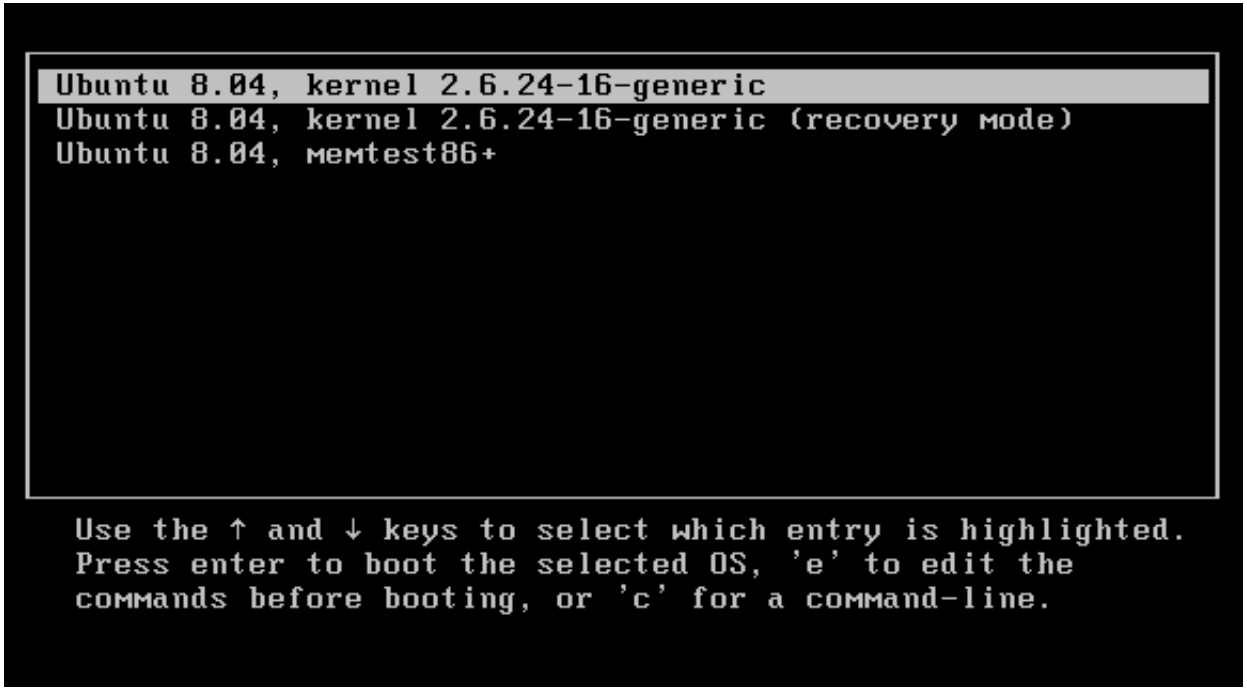
Dual boot example:

```
# fdisk xbd2
Disk: /dev/rxbd2d

Partitions aligned to 16065 sector boundaries, offset 63

Partition table:
0: NetBSD (sysid 169)
    bootmenu: NetBSD
    start 63, size 1044162 (510 MB, Cyls 0-64)
1: Linux extended (sysid 133)
    bootmenu: Linux
    start 1044225, size 1052927 (514 MB, Cyls 65-130/138/8)
2: <UNUSED>
3: <UNUSED>
Extended partition table:
E0: Linux native (sysid 131)
    start 1044288, size 530082 (259 MB, Cyls 65-97)
E1: Linux native (sysid 131)
    start 1574433, size 522719 (255 MB, Cyls 98-130/138/8)
Bootselector enabled, timeout 10 seconds.
First active partition: 0
Drive serial number: 0 (0x00000000)
#
```

Typical Boot Sequence



Typical Boot Sequence

- Power-on Self-Test
- primary boot loader (e.g. BIOS, UEFI, Open Firmware / OpenBoot)
- transfer of execution to Master Boot Record or perform netbooting
- Second-stage boot loader (e.g. GRUB)
- load kernel
- kernel transfers control to `init(8)`

Note: in virtualized environments, some of these steps are skipped, repeated, or simulated.

Typical Boot Sequence

- Power-on Self-Test
- primary boot loader (e.g. BIOS, UEFI, Open Firmware / OpenBoot)
- transfer of execution to Master Boot Record or perform netbooting
- Second-stage boot loader (e.g. GRUB)
- load kernel
- kernel transfers control to `init(8)`

Note: in virtualized environments, some of these steps are skipped, repeated, or simulated.

How do you know if your system is trustworthy once it boots up? Review "remote attestation" of software, "secure boot" mechanisms, and Trusted Computing.

Typical Boot Sequences

Exercise:

`https://stevens.netmeister.org/615/boot-sequence/`

```
$ aws ec2 run-instances --instance-type t1.micro --image-id ami-569ed93c
$ id=$(aws ec2 describe-instances --query 'Reservations[].Instances[].InstanceId')
$ aws ec2 get-console-output --instance-id ${id} | more
```

Compare the dislabel on the `/boot` device to the output of `df(1)` on the mounted partition. What's different?

Review the full console output; pay attention to the filesystem specific parts. Can you explain what's happening?

HW and Reading

`https://stevens.netmeister.org/615/s20-hw2.html`

- as always: manual pages for all commands
- `fs(5)`, `newfs(8)`, `tunefs(8)`
- `https://wiki.osdev.org/Partition_Table`
- `https://en.wikipedia.org/wiki/Master_boot_record`
- `https://is.gd/8KHnQj`
- `https://is.gd/wGgJ0e`

Additional Content

Content for topics we couldn't fit into class below.
Please review on your own time.

Basic Filesystem Concepts: The UNIX Filesystem

File types:

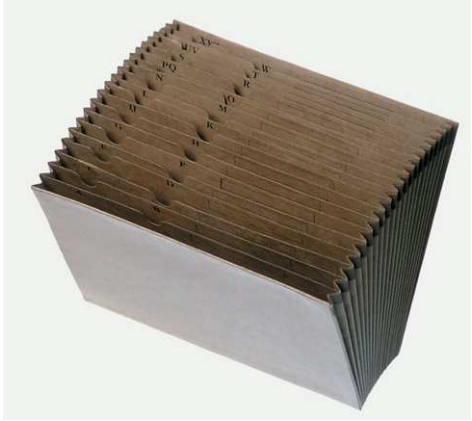
- regular files

```
$ stat /etc/passwd
```

Basic Filesystem Concepts: The UNIX Filesystem

File types:

- regular files
- directories



```
$ stat /
```

Basic Filesystem Concepts: The UNIX Filesystem

File types:

- regular files
- directories
- special files

```
$ file /dev/* | more
```

Basic Filesystem Concepts: The UNIX Filesystem

File types:

- regular files
- directories
- special files
- links

```
$ touch /tmp/foo
$ ln /tmp/foo /tmp/bar
$ stat /tmp/foo /tmp/bar
$ ln -sf /tmp/foo /tmp/bar
$ stat /tmp/foo /tmp/bar
```

Basic Filesystem Concepts: The UNIX Filesystem

File types:

- regular files
- directories
- special files
- links
- sockets

```
$ stat /dev/log
```

Basic Filesystem Concepts: The UNIX Filesystem

File types:

- regular files
- directories
- special files
- links
- sockets
- named pipes

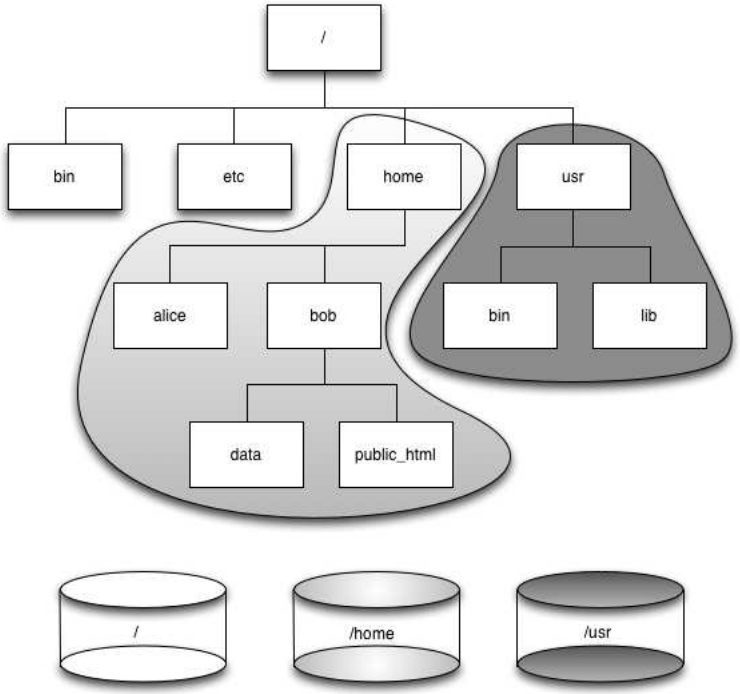
```
$ mkfifo /tmp/fifo
$ cat /tmp/fifo > /tmp/out &
$ stat /tmp/fifo | tee /tmp/fifo
$ cat /tmp/out
```

Basic Filesystem Concepts

Filesystem Layout

Basic Filesystem Concepts

All partitions – with the exception of the *root* (or /) partition – can be *mounted* anywhere in the filesystem hierarchy.



Basic Filesystem Concepts

All partitions – with the exception of the *root* (or `/`) partition – can be *mounted* anywhere in the filesystem hierarchy.

The file `/etc/fstab` (see fstab(5)) specifies which disks / partitions to mount where:

```
/dev/xbd1a /         ffs    rw 1 1
/dev/xbd0a /grub     ext2fs rw 2 2
kernfs     /kern     kernfs rw
ptyfs      /dev/pts  ptyfs  rw
procfs     /proc     procfs rw
```

Basic Filesystem Concepts

All partitions – with the exception of the *root* (or `/`) partition – can be *mounted* anywhere in the filesystem hierarchy.

The file `/etc/fstab` (see fstab(5)) specifies which disks / partitions to mount where:

```
$ cat /etc/fstab
# /etc/fstab: static file system information.
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
/dev/mapper/nemo--vg-root /               ext4    errors=remount-ro 0       1
# /boot was on /dev/sda1 during installation
UUID=e5abfdb9-1d53-4df7-9894-1c5a08b7f8c7 /boot           ext2    defaults        0
/dev/mapper/nemo--vg-swap_1 none            swap    sw              0       0
```

Basic Filesystem Concepts

All partitions – with the exception of the *root* (or `/`) partition – can be *mounted* anywhere in the filesystem hierarchy.

To see what filesystems are currently mounted, run `mount(8)`:

```
/dev/xbd1a on / type ffs (local)
/dev/xbd0a on /grub type ext2fs (local)
kernfs on /kern type kernfs (local)
ptyfs on /dev/pts type ptyfs (local)
procfs on /proc type procfs (local)
/dev/xbd2a on /mnt type ffs (local)
```

Basic Filesystem Concepts

```
$ mount
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=1002928k,nr_inodes=250732,mode=75
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=204804k,mode=755)
/dev/mapper/nemo--vg-root on / type ext4 (rw,relatime,errors=remount-ro,data=ordered)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
/dev/vda1 on /boot type ext2 (rw,relatime,block_validity,barrier,user_xattr,acl)
ldap:nirvana.phy.stevens-tech.edu:ou=auto.home,dc=phy,dc=stevens-tech,dc=edu on /home
kronos.srcit.stevens-tech.edu:/xraid0-1/export/home/jschauma on /home/jschauma type n
```

Basic Filesystem Concepts

Some of the different kinds of filesystems:

- “Regular” File Systems
- Journaling File Systems
- Network File Systems
- Various

Where do we put all of our files?

Layout of filesystem *should* be standardized. Some UNIX versions adhere to these standards, some are strongly influenced by tradition.

```
man hier
```

File System Hierarchy

```
/          root directory of the system

/bin/      utilities used in both single and multi-user environments

/dev/      block, character and other special device files

/etc/      system configuration files and scripts

/lib/      dynamic linked libraries used by dynamic linked programs (such
           as those in /bin/ and /sbin/) that cannot rely upon /usr/lib/
           being available.

/sbin/     system programs and administration utilities used in both sin-
           gle-user and multi-user environments

/tmp/      temporary files, usually a mfs(8) memory-based filesystem (the
           contents of /tmp are usually not preserved across a system
           reboot)

/usr/      contains the majority of the system utilities and files


           bin/      common utilities, programming tools, and applica-
                     tions

           lib/      archive, profiled, position independent archive, and
                     shared libraries

           sbin/     system daemons and system utilities (normally exe-
                     cuted by the super-user)

           share/    architecture-independent text files
```