1: A kernel overview

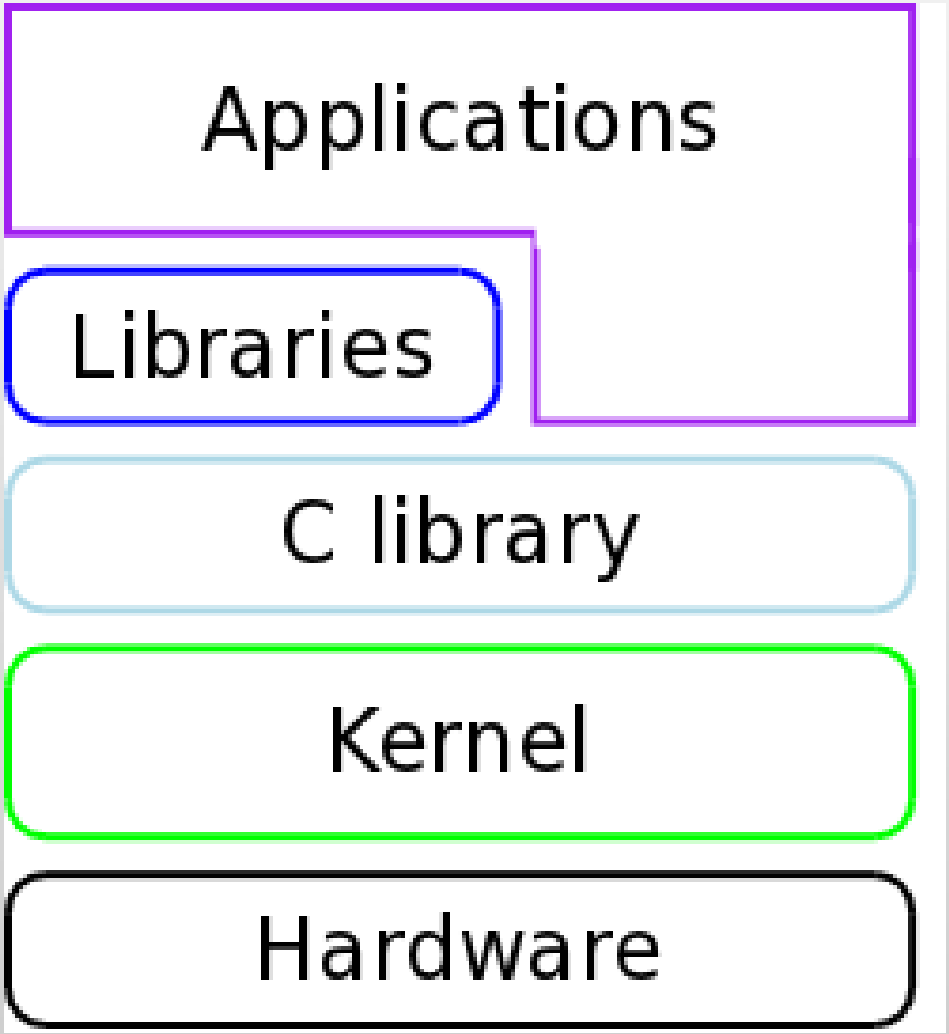
Good morning!

The plan for the day

A quick overview of kernel development
Process management
Low-level memory management
Process memory management
The virtual filesystem
Locking

The kernel

The core of the operating system



The kernel

The kernel handles

- Device management and abstraction
- Memory management
- Resource scheduling
- Process management
- Network communications
- Security
- ...

Why the kernel matters

You cannot work around it

- The kernel limits performance
- It limits features

It shapes how the system is programmed

Some rules for kernel development

This discussion is vague and handwavy

It's important, though:

- Much of what happens in kernel development follows from these ideas

We'll get technical soon, I promise

1: Upstream first

Code goes into the mainline first

- Before shipping to customers
- Before user space depends on it
- Before it's too late to change it

1: Upstream first

Example: Android wakelocks

- User-space API not mergeable
- Extensive changes needed in general
- Merging of drivers held up

2: No differentiation

Originally expressed by Andrew Morton

See “upstream first”

2: No differentiation?

“The RHEL6 kernel includes numerous subsystems and enhancements from 2.6.34, as well as its predecessor versions. As a result, the RHEL6 kernel cannot be simply labeled as any particular upstream version. Rather, the RHEL6 kernel is a hybrid of the latest several kernel versions.”

-- Red Hat Enterprise Linux Team

2: No differentiation

Example: out-of-tree drivers in Ubuntu

- Developers are upset
- Does not help the kernel progress
- Potential copyright issues

...they are moving away from this practice

3: Technical quality

Code quality outweighs everything else

- Company plans
- Users desires
- Existing practice
- Developer status
- Who got there first
- ...

3: Technical quality

Examples

- Device Mapper and EVMS
- Perf and perfmon2
- Schedulers: O(1), Fair sched, CFS
- devfs

3: Technical quality

How is quality measured?

- Cleanness
- Generality (multiple users)
- Size and performance
- Documentation
- Developer reputation

4: Peer review

No code is so good it can't benefit from another set of eyes

4: Peer review

Corollaries:

Trying to merge unreviewed code is a mistake

Ignoring review comments is a good way to keep your code out of the kernel.

5: Long-term view

We'll still be working on the kernel 5-10 years from now

5: Long-term view

The maintenance cost of every change will be evaluated

5: Long-term view

Corollary: no internal API stability

> “In Linux, we've rewritten our USB stack three or four times. Windows has done the same thing, but they had to keep their old USB stack and a lot of their old codes in order to work for those old drivers. So their maintenance burden goes up over time while ours doesn't.”

-- Greg Kroah-Hartman

5: Long-term view

Corollary: user-space ABI additions will be scrutinized closely

- They have to be supported forever!

6: No regressions

...not even to fix other problems

> “So we don't fix bugs by introducing new problems. That way lies madness, and nobody ever knows if you actually make any real progress at all. Is it two steps forwards, one step back, or one step forward and two steps back?”

--Linus Torvalds

7: Code talks

“Talk about high level designs rarely gets any traction, and often goes nowhere. Give us an example implementation so there is something concrete for us to sink our teeth into.”

-- David Miller

8: No ownership of code

Free software means giving up control.

9: Developers are individuals

...separate from their employers

10: Kernel development should be fun

Getting the kernel

ftp.kernel.org (pub/linux/kernel/v2.6)

Git

git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git

Distributor source packages

The kernel code tree

2,190 directories
33,209 files
9,187,929 lines of code

(2.6.37-rc3)

The kernel source tree

Lines	Subdirectory
5,266,304	drivers/
1,774,426	arch/
670,346	fs/
479,641	sound/
452,233	net/
271,134	include/
110,814	kernel/
49,635	mm/
...	

Licensing

Kernel code carries a variety of licenses

- GPLv2
- GPLv2+
- BSD
- public domain
- ...

The kernel as a whole is GPLv2

Licensing notes

Distribution of kernel code must be done according to the terms of the license

Licensing notes

The kernel has no copyright assignment requirement

==> Thousands of copyright owners

Licensing notes

A change of license is highly unlikely

The DCO

Developers certificate of origin

1) I have the right to contribute this code
2) The kernel project can store my info

See Documentation/SubmittingPatches

In the beginning

...the computer is without form or guidance.

Then the bootloader starts

The bootloader's job

Minimal hardware configuration

Load the system

- Compressed kernel image
- Initial ramdisk (if any)

Jump into the loaded kernel

Early boot

Perform some memory setup

Uncompress the kernel into place

A typical memory layout

```
00000000-00000fff : reserved
00001000-0009fbff : System RAM
0009fc00-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000cee00-000cffff : pnp 00:0d
000d0000-000d0fff : Adapter ROM
000d1000-000d1fff : Adapter ROM
000d2000-000d3fff : pnp 00:0d
000e0000-000fffff : reserved
  000f0000-000fffff : System ROM
00100000-7e7affff : System RAM
  00400000-007acacd : Kernel code
  007acace-00a1cf6f : Kernel data
  00aa8000-00c2b8bf : Kernel bss
```

Bootstrap continues

The uncompressed kernel runs

- IRQs disabled
- Initialize data structures
- Initialize scheduler
- Turn on slab allocator
- Connect to console
- Create init and kthreadd tasks
- Turn on the scheduler

The init task

Init's job:

- Start all other CPUs
- Complete hardware initialization
- Delete bootstrap code
- Run ramdisk init (if any)
  - That init should call pivot_root()
- Exec the real init program
  - init, upstart, systemd, ...

At this point

There are two processes running
(init, kthreadd)

...maybe it's time to talk about processes...

Questions?