A quick glance at the Linux Kernel

by
Manu Abraham

Operating System

An Operating System consists of the kernel and utilities

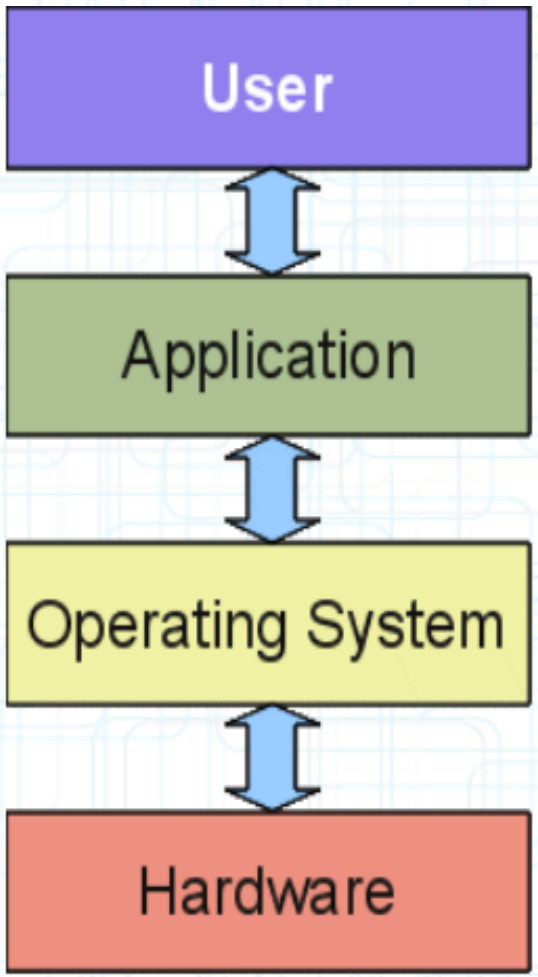
The kernel is the most important part of an OS.

Provides programs with a consistent view of the hardware.

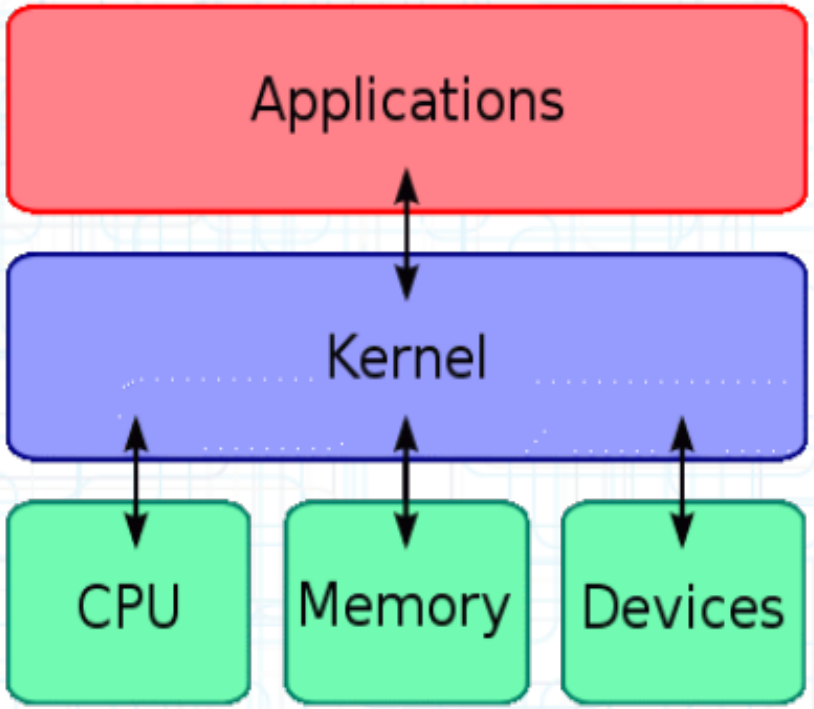
OS protection achieved by splitting execution modes.

Kernel: privileged mode

User:unprivileged mode



Kernel, the black box



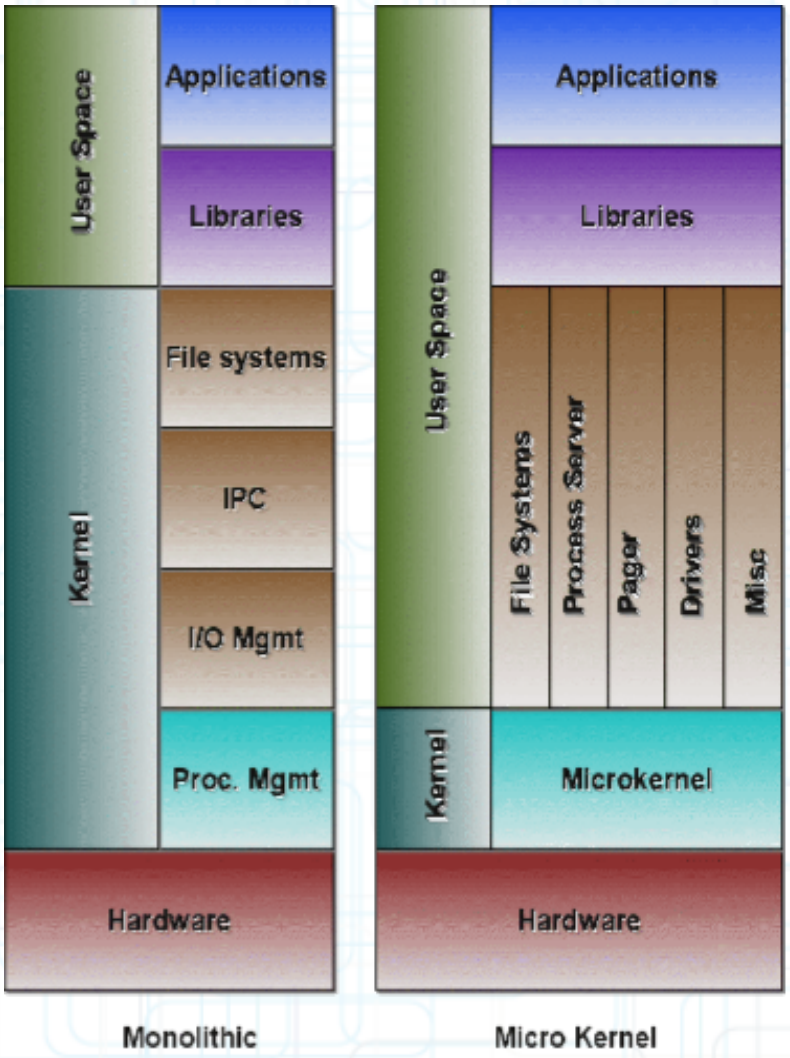
Process Management

Memory Management

Device Management

System Calls

Kernel types



Monolithic

Memory management in kernel space

Recompile kernel with any change

Large footprint

Micro-kernel

Leaner kernel

Easily extensible

Hard to implement

Other Kernel types

Hybrid Kernels – a compromise

Nano Kernels – delegate all services

Exo kernels – no abstraction of hardware

The Linux Kernel

A monolithic kernel system like UNIX

Kernel support for loadable modules

Loadable module support through user applications such as insmod, rmmod etc

All code covered under the GPL v2.

Extensive use of the C programming language

Architecture specific code lives under arch/

Support for a wide range of architectures.

Applications write-once, run on multiple architectures, with no overhead.

Concurrency

Preemptible kernel

Symmetric multiprocessor capable

Kernel code is reentrant

Kernel Versioning

Releases with version 2.6.x

Release cycles 2 – 3 months

Managed by Linus Torvalds

Maintainer: Andrew Morton

Stable releases are labeled 2.6.x.y

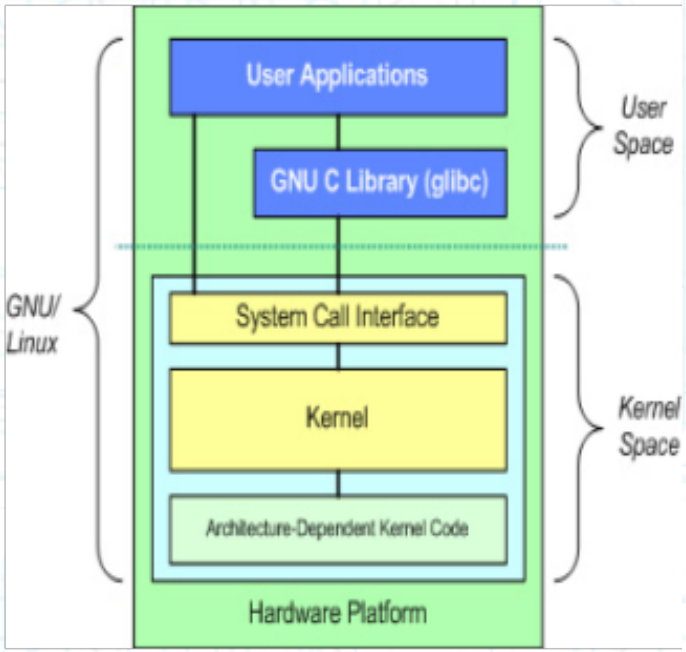
Release after a 2.6.x

Managed by Greg Kroah Hartman

2.4.x kernels are obsolete

So are the Even/Odd release cycles

Linux Architecture



GNU C Library provides the system call interface that connects to the kernel

kernel and user application occupy different protected address spaces

the kernel occupies a single address space

Much to the UNIX philosophy everything's a file

glibc

GNU C library commonly called glibc

Released by the GNU project

http://gnu.org/software/libc

Free software released under LGPL

Provides functionality for SUS, POSIX, SVID, X/Open

A very large codebase/footprint

For embedded applications leaner uClibc proves to be a better choice. http://www.uclibc.org

System Call Interface

Kernel code in Ring 0 is invoked from Ring 2/3

Traditionally, user copies syscall id to CPU register EAX and execute INT80

This proved to be a slow approach

Linux 2.6 takes advantage of SYSENTER/EXIT CPU (Fast System Call) instructions

CPU can do a Fast system call ? Check CPUID register Bit: 11

ftp://download.intel.com/design/pentiumii/manuals/24319102.pdf

Kernel Development

A community development model.

Different communities for different subsystems

C programming knowledge a necessity.

Adheres to very strict methodologies, abstracted from various styles

Code merged into mainline is available to all Linux users.

One source for all Linux distributions

Internal kernel API is in constant flux, a deliberate decision.

Code subjected to review and hence better quality

Building a Linux Kernel

Download the kernel source

http://www.kernel.org/pub/linux/kernel/v2.6/

Choose a version; eg: linux-2.6.37.tar.bz2

Install kernel source in /usr/src/linux/.

Configure a Kbuild

Start with an initial Kconfig

Customize your Kconfig eg: make menuconfig

Build the kernel: eg: make -j4

Install kernel : eg: make modules_install; make install

Looking at Kernel messages

dmesg – prints the kernel message ringbuffer.

/var/log/messages – system log files from syslog

Licensing

All code contributed to the Linux kernel has to be legitimately free software.

Code that is contributed to the Linux kernel has to be compatible with version 2 of the GPL

Dual licensing using the three-clause BSD license is not uncommon.

Copy of GPL v2 can be obtained from http://www.gnu.org/licenses/gpl-2.0.html

Binary Modules

Binary modules greatly increase the difficulty of debugging kernel problems.

Legal issues surrounding the distribution of proprietary kernel modules, general thought follows that it is a violation of the GPL.

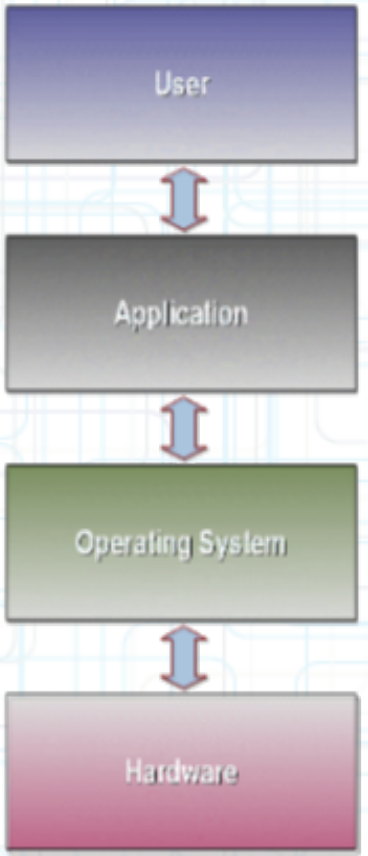
Assuring compatibility between different kernel versions becomes a nightmare.

What is a device driver ?

Device driver are like “black boxes” that make a particular hardware respond to a defined programming interface.

User interactions perform standardized calls, that are independent of the driver.

A software layer that exists between the applications and the physical device.



Development Tools

GNU Tools collection. eg: gcc, make ..

LXR (linux cross reference) project http://lxr.linux.no

Usermode Linux is useful in some cases

Git for managing changesets

Driver Types

Kernel Mode

Also known as Kernel space driver

Most Linux drivers are Kernel Mode

User Mode

Also known as User space driver

Best example is the Xorg driver

Kernel Mode driver

Drivers are modular. Bye, monolithic builds.

Runs in Kernel space.

Need to be loaded separately.

Manual loading

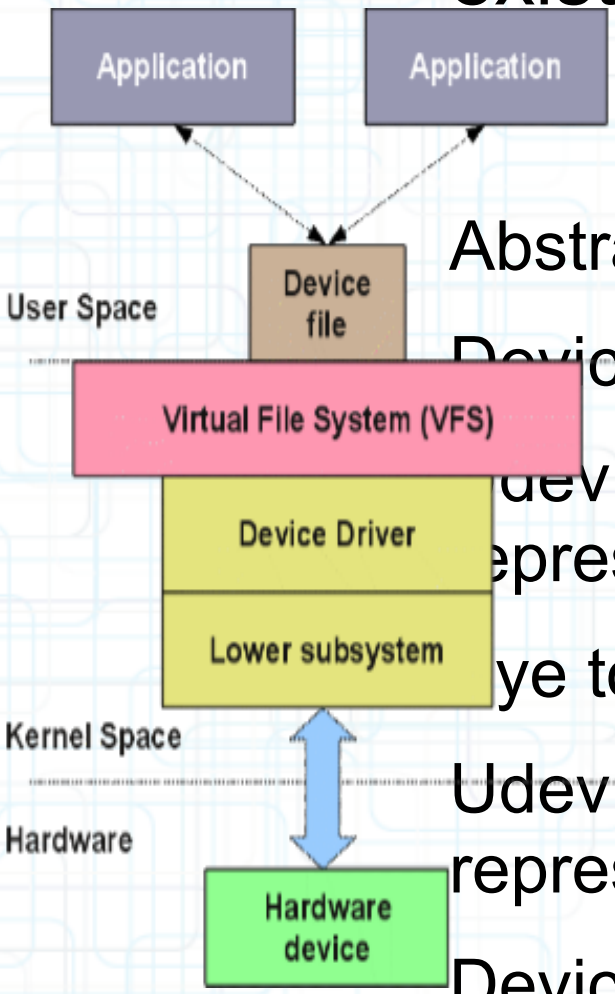
Automatic loading

User application is required to load driver.

Powerful and fast, since it has access to complete system resources.

Programming style slightly different from User space programming.

A generic device abstraction exists.



Abstraction accessible through VFS.

Device tree info exported via sysfs.

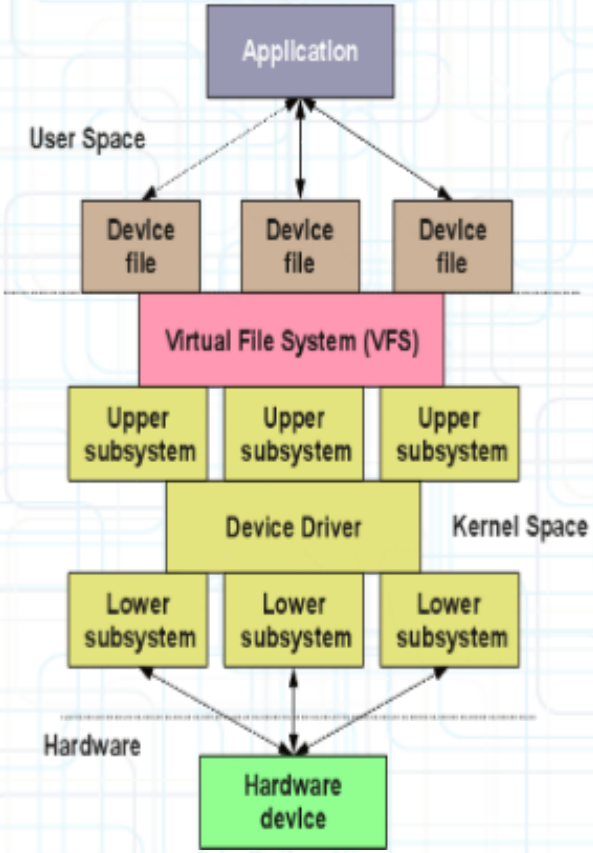
udev uses sysfs info and a set of rules to represent the device in /dev

Bye to MKNOD, with udev

Udev requires rules for device representation

Device file referred to by <major, minor> pair.

Layered Drivers



Driver registers with multiple subsystems involved

Subsystem - Code shared by drivers

Can be providing hardware interface logic, a protocol, or an interface logic

Life made easier, driver can concentrate on specific driver functionality alone

Adding new functionality to a subsystem could be hard

Linux subsystems

ALSA (Advanced Linux Sound Architecture)

www.alsa-project.org

alsa-announce@alsa-project.org

alsa-devel@alsa-project.org

DVB (Digital Video Broadcasting)

www.linuxtv.org

linux-dvb@linuxtv.org

linux-media@vger.kernel.org

I2C (Inter IC Communication)

linux-i2c@vger.kernel.org

Network devices

netdev@vger.kernel.org

PCI (Peripheral Component Interconnect)

linux-pci@vger.kernel.org

SCSI (Small Computer System Interface)

linux-scsi@vger.kernel.org

USB (Universal Serial Bus)

linux-usb@vger.kernel.org

V4L (Video for Linux)

www.linuxtv.org

linux-media@vger.kernel.org

Interfacing applications and drivers

Interfacing

Application: read(), write(), mmap(), ioctl()

Driver: copy_to_user(), copy_from_user(), mmap(), get_user_pages()

Determining User/Kernel split

No in-kernel processing

Device Types

Character devices

Accessed through filesystem names

Special files located in /dev

read(), write()

Block devices

Special files located in /dev

block_read(), block_write()

Network devices

devices not visible in /dev

Normal file ops do not make sense

User Mode driver

A generic driver is not needed

mmap() a possibility

I/O space access with inb(), outb()

Doesn't work well with interrupts

Easier to debug and add functionality

UIO a minimal stub in kernel

eg: libusb, X, directfb

CodingStyle

For any code to be accepted into the mainline kernel tree, it has to follow the CodingStyle guidelines.

CodingStyle guide can be obtained from Documentation/CodingStyle

CodingStyle exists to improve code readability for the community members, resulting in better code quality.

Style guidelines

Indentation: Tabs are 8 chars

Whitespaces: cause merge issues

Braces: K&R style

Naming: keep it short

Typedefs: avoid opaqueness

Functions: keep them short

Comments: C89 style, don't use C99 style

Macros: don't reinvent them

Verify style issues with scripts/checkpatch.pl

Posting patches

Always test code before posting

Make sure it confirms to CodingStyle

Patches that modify performance behaviour need to show benchmark results.

Make sure you have the right to publish the code.

“From:” line naming the patch author. Eg: From: myemail@mydomain.com

One line description of what the patch does. Eg: stv090x: improve tuning speed

A blank line followed by a detailed description of the patch.

One or more Tags, a minimum of a Signed-off-by: line from the author of the patch.

The patch should be in the unified format -u

Post shall be copied to

maintainer(s) of affected subsystem(s)

Other developers working in the same area.

Relevant subsystem mailing-list and or linux-kernel@vger.kernel.org

Original poster, in response to a bug/feature

Version Control System (VCS)

Centralized model (CVS, SVN ..)

Small project

Distributed model (Git, Mercurial ..)

Large project scattered globally

Linux kernel development uses git

Hosted at git.kernel.org

Master branch merges by Torvalds.

Thank You!

Thank you for listening

Feel free to contact me

Manu Abraham

Manu (AT) linuxtv (DOT) org

Manu (AT) kernel(DOT)org
Abraham(DOT)manu@gmail (DOT) com

or |MA| on irc.freenode.net #dubailug