Introduction to the Linux kernel: challenges and case studies

Juan Carlos Sáez Alcaide

Department of Computer Architecture and Automation
ArTeCS Group
Complutense University of Madrid

IV Semana de la Informática 2018

Feb 8, 2018

About Me

- **Juan Carlos Sáez Alcaide** (jcsaezal@ucm.es)
  - Interim Associate Professor, UCM
  - Department of Computer Architecture and Automation
- Teaching: *Operating Systems*, *Linux and Android Internals*, ...
- Member of the ArTeCS Research Group
  - High Performance Computing
  - Computer Architecture
  - Interaction between system software and architecture
  - ...
- UCM Campus Representative of the USENIX Int'l Association
  - Login (USENIX Magazine)

Outline

Outline

Unix (I)

- Unics – Unix (1969)
  - Created by Ken Thompson and rewritten in "C" by Dennis Ritchie (1973)
  - V6 (1975): Public source code (AT&T license)
    - BSD distributions (Billy Joy)
    - John Lion's book on UNIX V6

Keys to success

1. Inexpensive license
2. Source code available
3. Code was simple and easy to modify
4. Ran on modest HW

Unix (II)

- Unix (Cont.)
  - V7 (1979): code can be no longer used for academic purposes
- Xenix (1980)
  - Microsoft
  - SCO
- Unix System III (1982)
- Unix System V (1983)
  - HP-UX, IBM's AIX, Sun's Solaris

Unix (III)

- Proyecto GNU (1983) - Richard Stallman
  - SO GNU: Emacs, GNU compiler collection (GCC), GNU Hurd (kernel)

Richard Stallman

- Minix v1 (1987) - Andrew Tanenbaum
  - Minimal Unix-like OS (Unix clone)
  - Teaching purposes. Modular structure
  - Compatible with Unix V7 (user level) evolved towards the POSIX standard
    - 1987 (Minix1 - i8088), 1997 (Minix2 - i386)

Andrew Tanembaum

Conflicts in the Unix world

Unix Wars (1987-1996)

- Unix International (USL - AT&T) vs. Open Software Foundation
  - Unix System V Release 4 (SVR4)
- War of specifications
- War ends in 1996 $\rightarrow$ Open Group (Unix Trademark)

Lawsuit brought by USL-AT&T against BSDI (1991-1994)

- Issue: attempt to make BSD free of any UNIX code
- Agreement is reached in 1994: Novell acquires USL

And Linux comes out... (1991)

- A few months after Minix 1 was released the following message was posted at **comp.os.minix**:

```
From: torva...@klaava.Helsinki.FI (Linus Benedict Torvalds)
Date: 25 Aug 91 20:57:08 GMT Local: Sun, Aug 25 1991 9:57 pm
Subject: What would you like to see most in minix?

Hello everybody out there using minix - I'm doing a (free) operating
system (just a hobby, won't be big and professional like gnu) for
386(486) AT clones. This has been brewing since april, and is starting
to get ready. I'd like any feedback on things people like/dislike
in minix, as my OS resembles it somewhat (same physical layout of
the file-system (due to practical reasons) among other things).
I've currently ported bash(1.08) and gcc(1.40), and things seem to
work. This implies that I'll get something practical within a few
months, and I'd like to know what features most people would want. Any
suggestions are welcome, but I won't promise I'll implement them :)

Linus (torva...@kruuna.helsinki.fi)

PS. Yes it's free of any minix code and it has a multi-threaded fs. It
is NOT portable (uses 386 task switching, etc), and it probably never
support anything other than AT-harddisks, as that's all I have :(.
```

And Linux comes out... (1991)

- Keys to success
  - Unix Wars
  - BSD Lawsuit
  - Competition with Windows NT (common enemy)
  - GPL License
  - Internet
  - ???

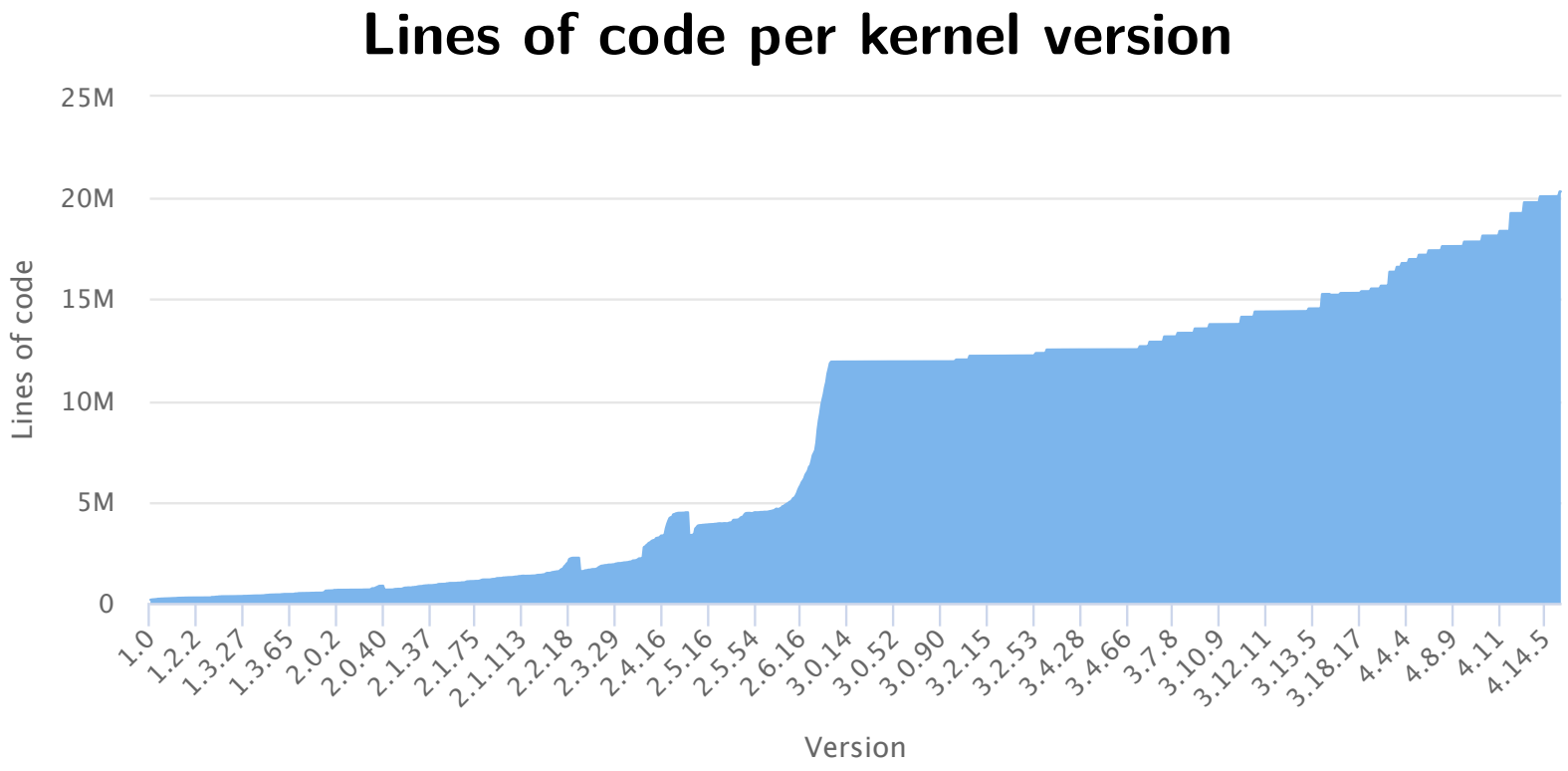
And Linux comes out... (1991)

- But Linux is just a kernel...
  - GNU/Linux..., IBM/RedHat/HP/....Linux..
- Distros (kernel + selection of pre-compiled tools)
  - MCC Interim Linux 1992,
  - Slackware (Patrick Volkerding) , Debian (Ian Murdock) 1993
  - S.U.S.E, Red Hat (Marc Erwing, Bob Young) 1994
- Desktop Environments
  - Xfree86 – Thomas Roel 1991
  - KDE – Matthias Ettrich 1996
  - Gnome – Miguel de Icaza 1997
  - ...

Linux Is Not UniX

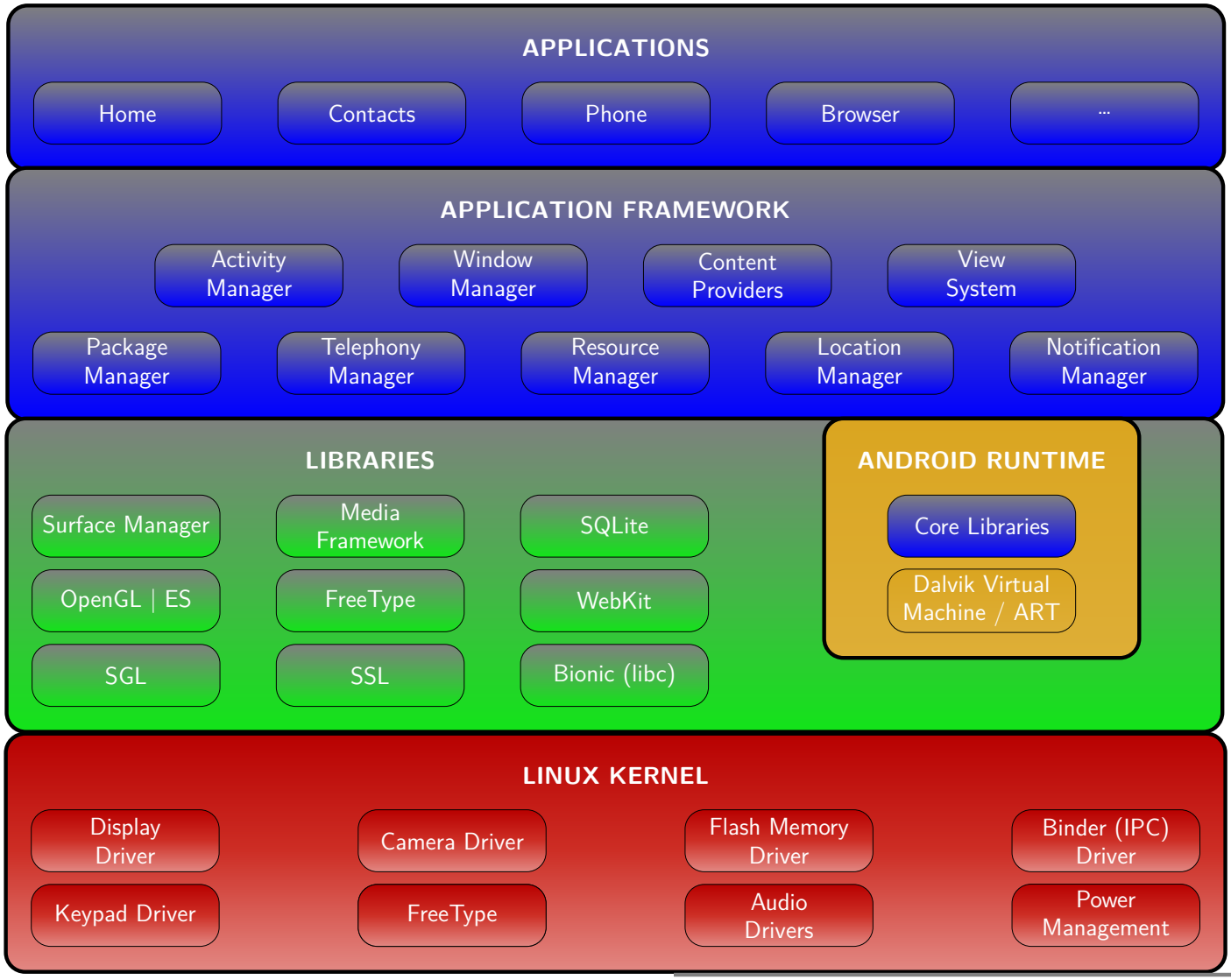
- Open Group
  - Current owner of the UNIX trademark
  - SUS (Single Unix Specification) certification
    - Unificación SUS / IEEE Posix en 2001 (SUS Version 3)
- *A Unix system must comply with SUS*
  - Mac OS X Leopard (based on BSD)
  - Z/OS IBM
- GNU/Linux is just a UNIX-like OS

Linux kernel: evolution since 1991



Source: https://www.linuxcounter.net

Android: a Linux-based OS

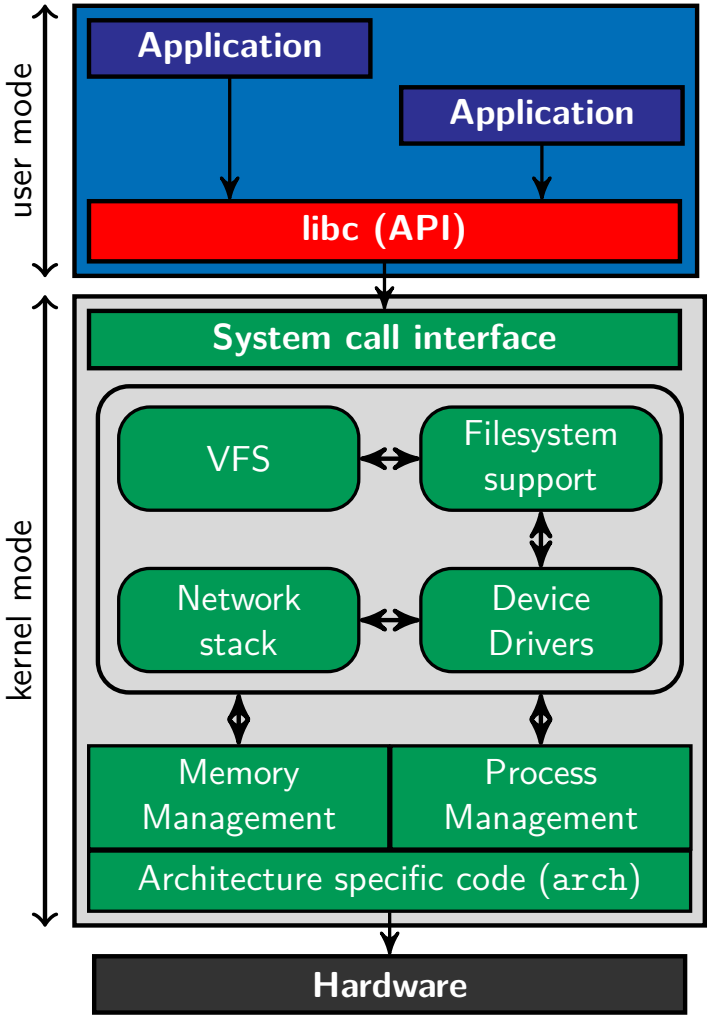


Linux kernel in Android

- Android relies on extensions made on top of the *vanilla* kernel:
  - Binder
  - Wakelocks
  - klogger
  - Ashmem
  - Low Memory Killer
  - Paranoid Network
  - Alarm Timers
  - ...
- Today, the "Androidized" kernel constitutes a significant *fork* of Linux
  - Linaro maintains an "androidized" kernel close to Linux *mainline*

Outline

Monolithic design



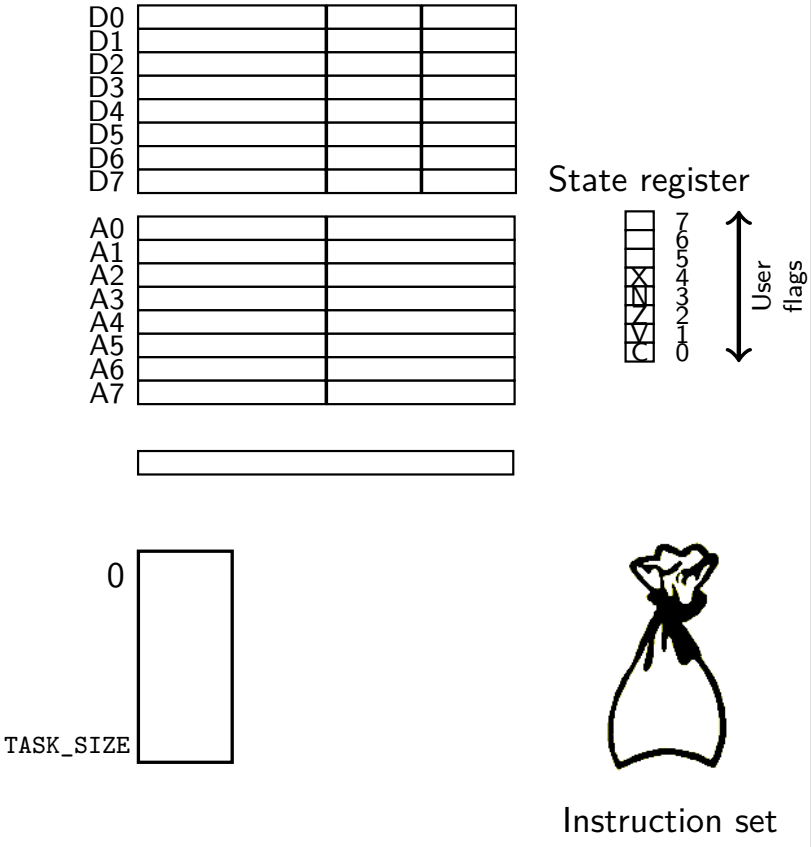
Features

- All kernel components share the same address space
- Everything runs in kernel (privileged) mode
- Good performance
- No isolation between components

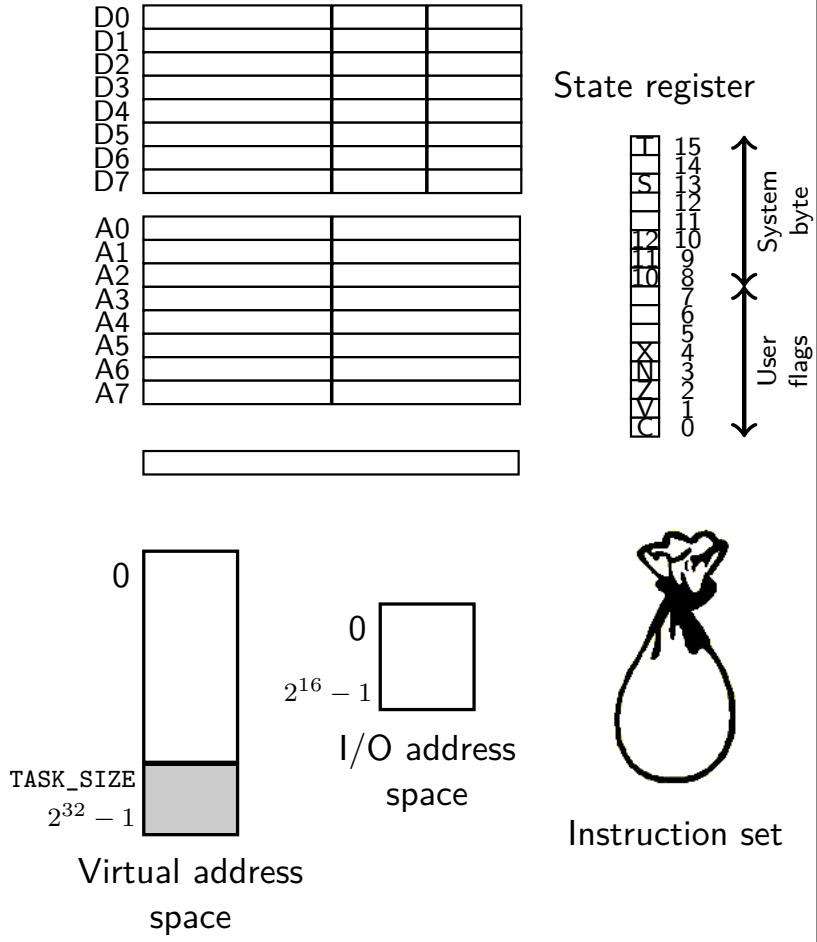
Kernel Components

User mode vs. kernel mode

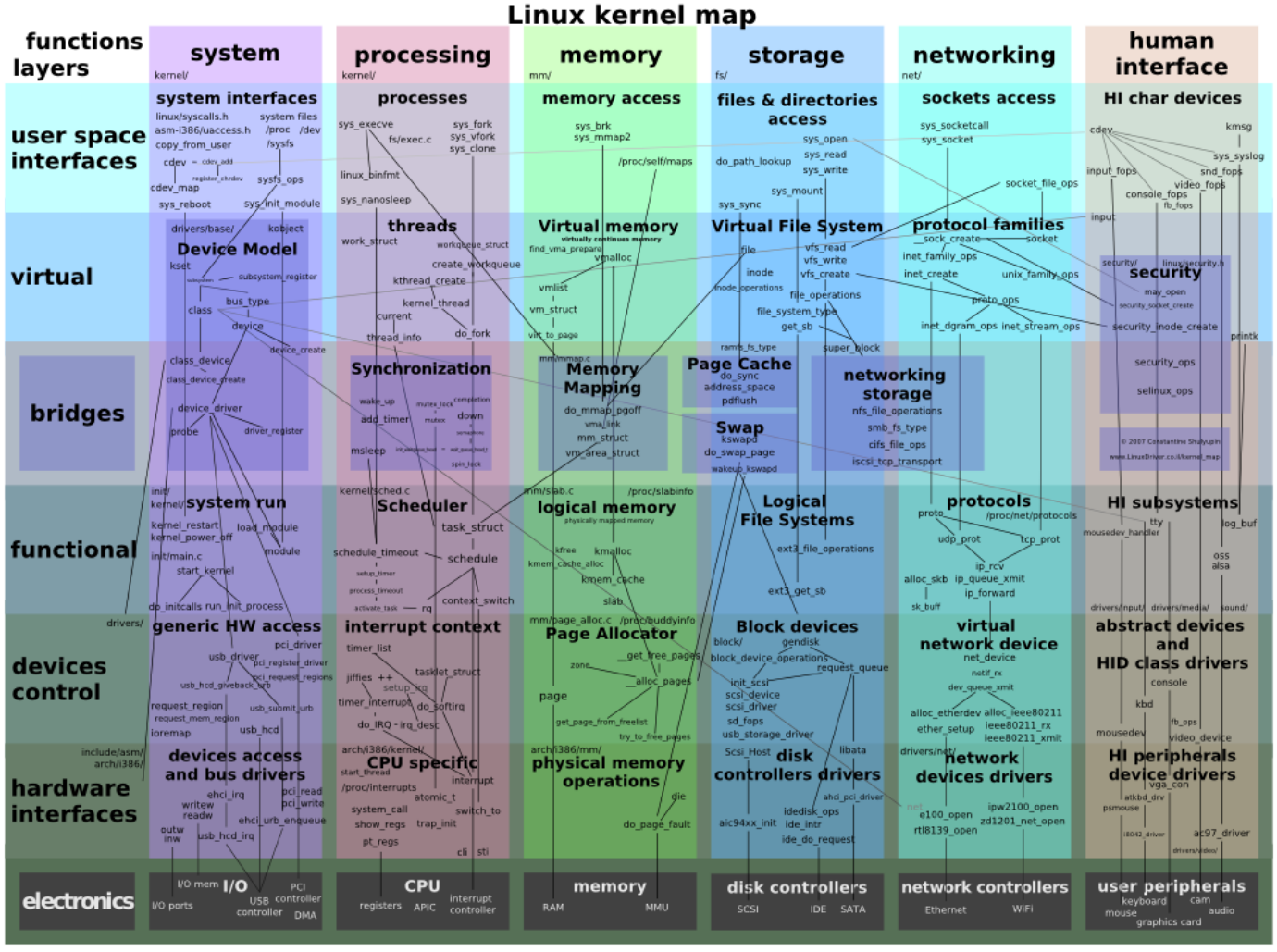
User mode



Kernel mode



Interactive map of the Linux kernel



http://www.makelinux.net/kernel_map/

Main challenges

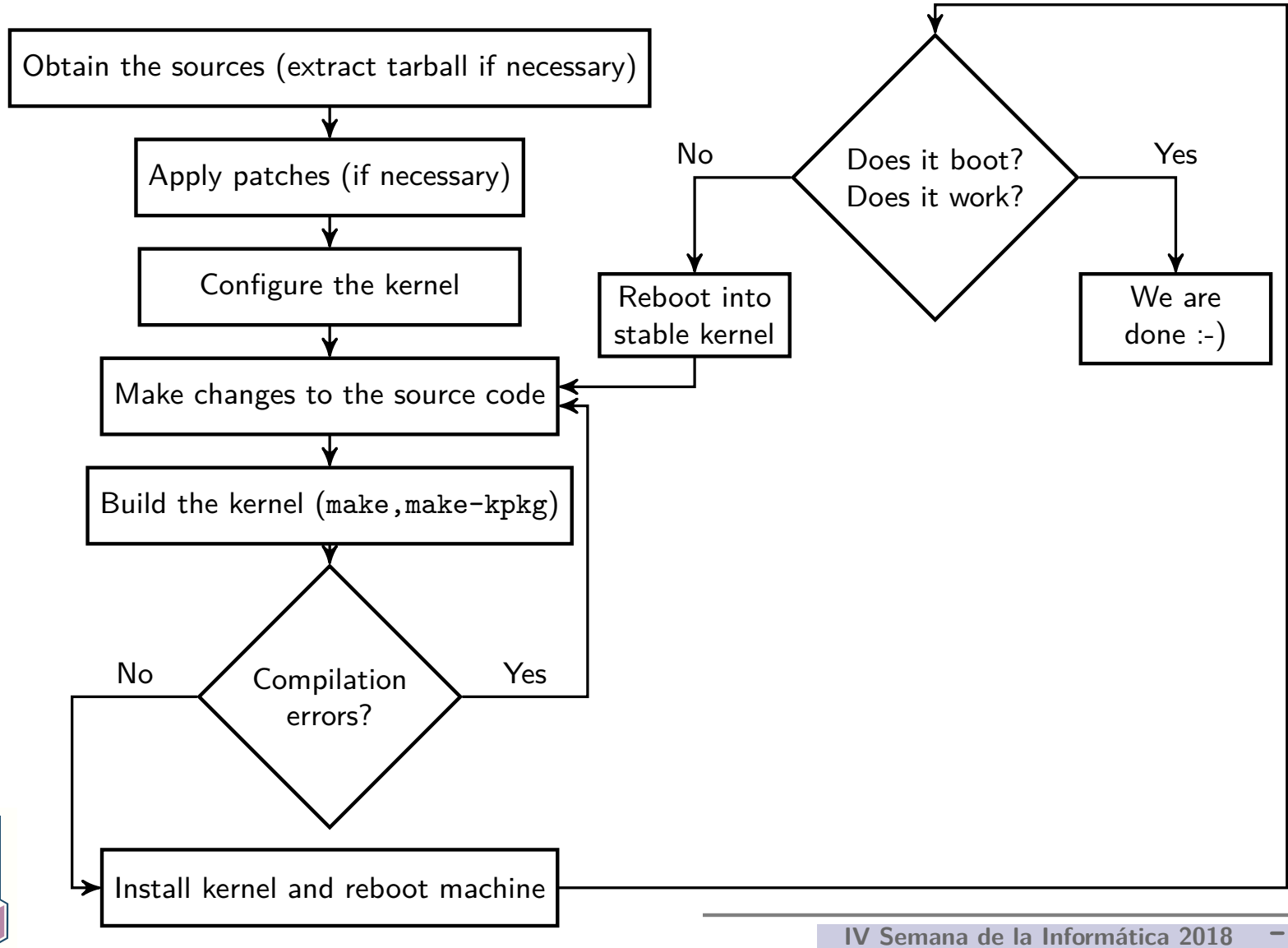
A beast of a different nature

- Development process is labor-intensive
  - Testing new features or bugfixes requires
    1. (Re)Build the kernel (it may take a while!)
    2. Install the new kernel
    3. Reboot the machine
- No memory protection (no SIGSEGV)
- User-space like debugging tools are not available
- Documentation becomes outdated quickly
  - *Developers must really understand the kernel code*
- No standard C library*
- Limited size of the kernel stack

Obtaining kernel sources

- Vanilla: (The Linux Kernel Archives) www.kernel.org (tarball or git repository)
  - `git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git`
  - `https://git.kernel.org`
- GNU/Linux distributions typically use Linux with patches
- Example Debian:
  - Package installation with `apt-get` o `apt source`
    - Source package of linux-image-*
    - linux-source .deb package (install tarball at /usr/src/)
  - More information at Debian Linux Kernel Handbook

Working with the Linux kernel



Linux kernel modules

What is a kernel module?

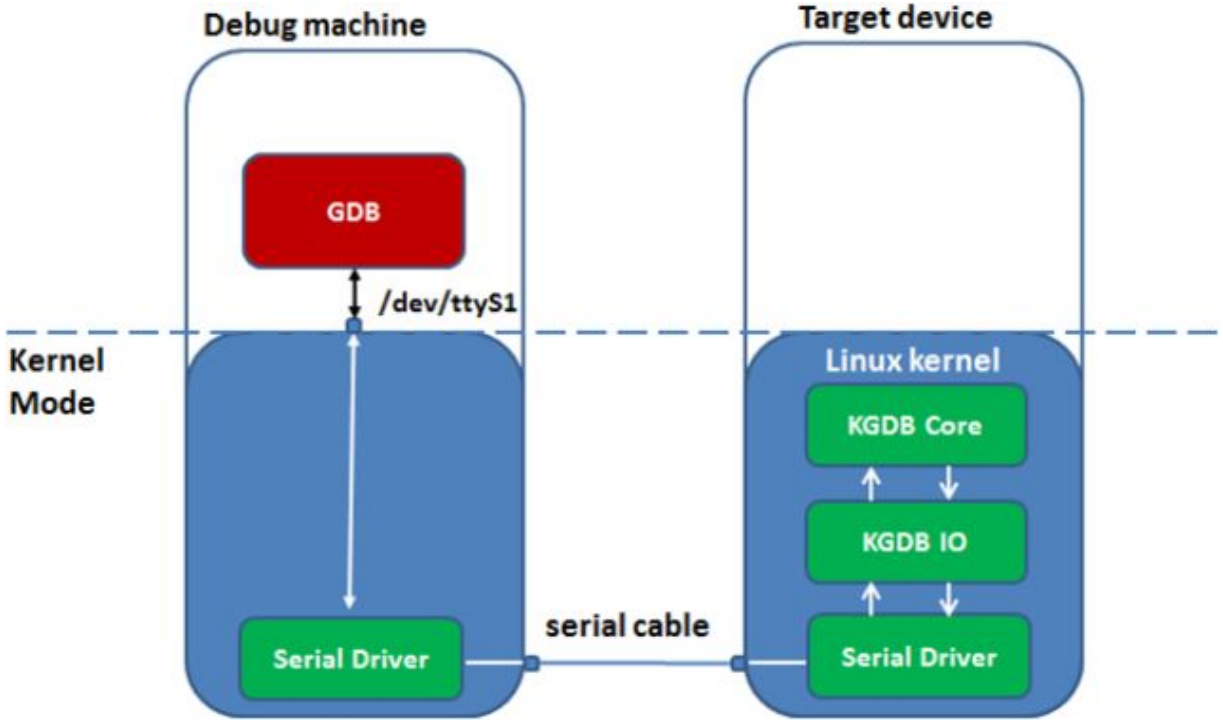
- A "code fragment" that can be loaded/unloaded into/from the OS kernel's address space on demand
  - Written in "C"
  - Bundled as a .ko file (object file)
    - Compilation driven by a Makefile
  - `insmod`, `rmmod`
- Its functions are executed in kernel mode

Limitation

- *Not everything can be implemented as a kernel module*
  - System calls
  - Scheduling algorithms
  - ...

KGDB

- Requires 2 machines connected via serial port
  - *Development host*: To build the kernel and invoke `gdb`
  - *Target system*: Runs the kernel with `KGDB` support



Source: https://blog.trendmicro.com

KDB

- KDB: in-kernel debug shell (serial port/text console)
  - No need to use an external debugging host
- KDB is not a source-level debugger
  - Programmer needs to deal with assembly code
  - You can use it in conjunction with gdb and external symbol file

terminal

```
$ echo g >/proc/sysrq-trigger
SysRq : DEBUG

Entering kdb (current=0xdfdff040, pid 71) due to Keyboard Entry
kdb> bp sys_sync+4
Instruction(i) BP #0 at 0xc00c9f00 (sys_sync+0x4)
   is enabled  addr at 00000000c00c9f00, hardtype=0 installed=0

kdb> go
$ sync

Entering kdb (current=0xdfdaa360, pid 72) due to Breakpoint @ 0xc00c9f00
kdb> bt
Stack traceback for pid 72
0xdfdaa360       72       71  1    0   R  0xdfdaa560 *sync
[<c0028cb4>] (unwind_backtrace+0x0/0xe4) from [<c0026d50>] (show_stack+0x10/0x14)
[<c0026d50>] (show_stack+0x10/0x14) from [<c0079e78>] (kdb_show_stack+0x58/0x80)
[<c0079e78>] (kdb_show_stack+0x58/0x80) from [<c0079f1c>] (kdb_bt1.clone.0+0x7c/0xcc)
[<c0079f1c>] (kdb_bt1.clone.0+0x7c/0xcc) from [<c007a240>] (kdb_bt+0x2d4/0x338)
...
```

SystemTap

- SystemTap: Tool for dynamic instrumentation of the kernel
  - Scripting language

example.stp

```
probe kernel.function("do_fork"){
        printf("do_fork() was invoked by PID=%d\n",pid());
}
```

terminal

```
$ sudo stap example.stp
do_fork() was invoked by PID=4811
do_fork() was invoked by PID=4820
do_fork() was invoked by PID=4811
...
```

Linux kernel debugging

- Many options available:
  - KGDB
  - KDB
  - GDB + `/proc/kcore`
  - Ftrace
  - SystemTap
  - ICE / JTAG (USB or ethernet)
  - printk() + dmesg
  - kdump/kexec
  - ...
- The challenge is knowing what to use when...
- Many tools have a steep learning curve

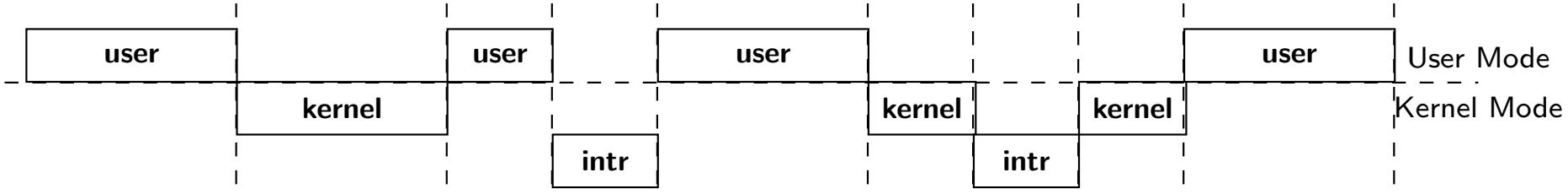
Outline

Kernel Control Paths

- Linux kernel is like a server that answers requests
  - Kernel functions are executed in response to *events*
    1. System calls
    2. An exception occurs (e.g., illegal instruction)
    3. An interrupt is raised by an I/O device
- **Kernel control path**: sequence of instructions executed in kernel mode
  - Lighter than a process (less context)
  - It does not always work on behalf of a process

Execution Contexts

- At a certain time instant t each CPU can be running:
  - A process's code at user space (**Process context in user mode**)
  - Kernel code from a system call or exception handler (**Process context in kernel mode**)
  - A *kernel thread*'s code (**Process context in kernel mode**)
  - Interrupt handling code (**Interrupt context**)

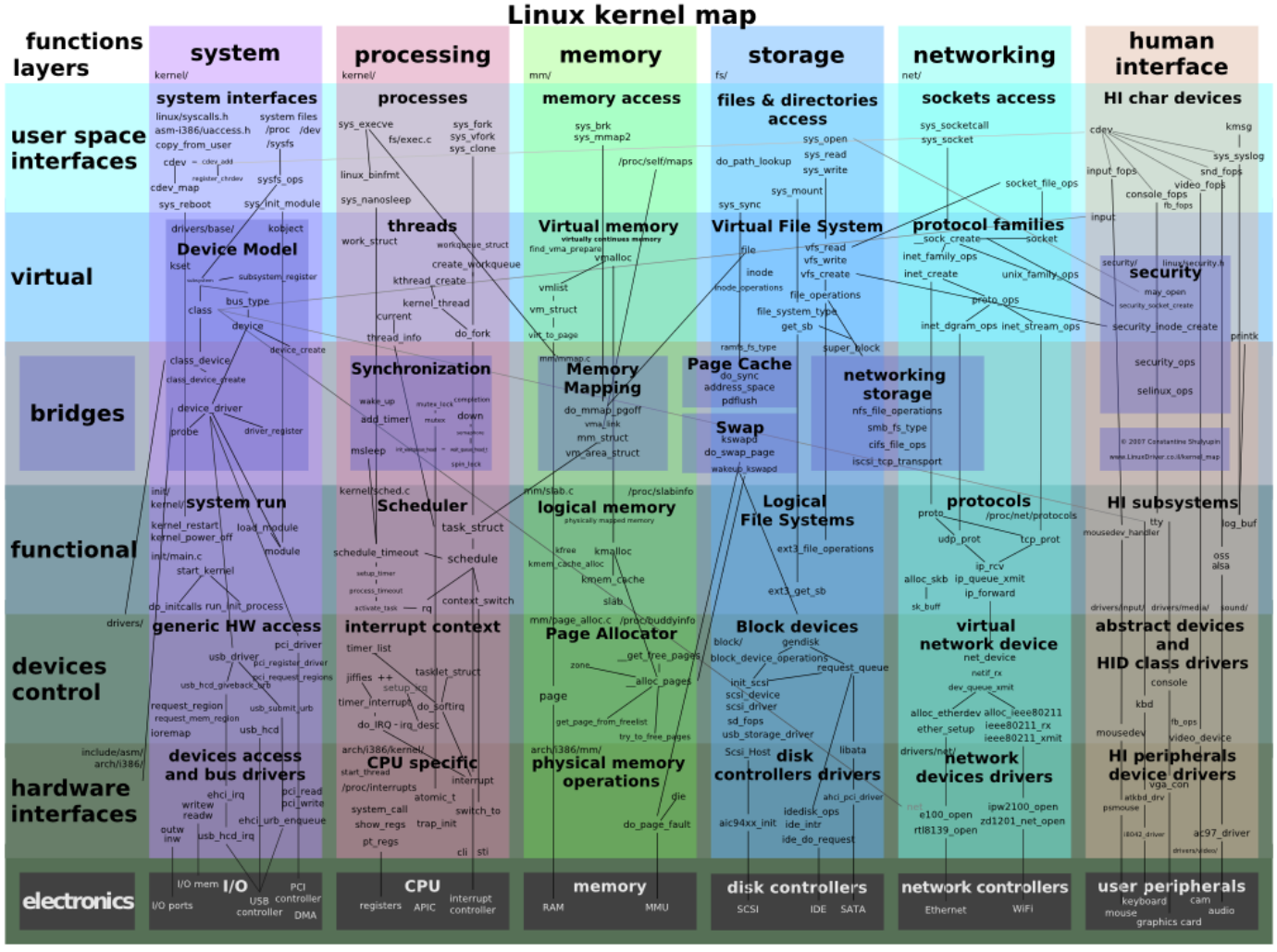


Causes of Concurrency

- Causes of interleaved/parallel execution of kernel code paths
  1. **Interrupts**
     - An interrupt can occur asynchronously at almost any time, inter- rupting the currently executing code
  2. **Bottom-half processing**
  3. **Kernel preemption**
     - Because the kernel is preemptive, one kernel code path can preempt another
  4. **Sleeping and synchronization with user-space**
     - A task in the kernel can sleep and thus invoke the scheduler, resulting in the running of a new process.
  5. **Symmetrical multiprocessing**
     - Two or more processors can execute kernel code at exactly the same time

Outline

Interactive map of the Linux kernel



http://www.makelinux.net/kernel_map/

Common Kernel abstractions

Common abstractions

- System calls
- Pseudo file systems: /proc, /sys
- Kernel data structures
- Dynamic memory allocation
  - `kmalloc()`, `vmalloc()`, `kfree()`, `vfree()`
- Bottom-half methods (deffering work)
- Kernel timers
- Kernel threads
- Kernel synchronization methods
- ...

Pseudo file systems

- Interaction between user programs and the kernel through files
  - We may read/modify kernel "variables" via shell commands
    - `echo 1 > /proc/sys/net/ipv4/ip_forward`
    - `cat /proc/cpuinfo`

/proc

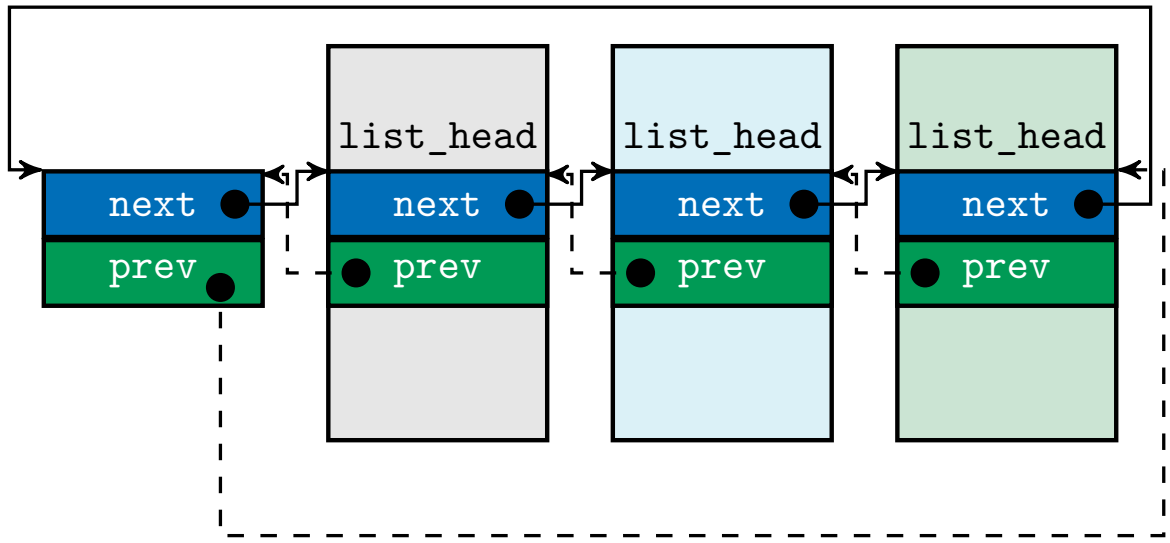
- Programmer must provide an implementation of the various file operations (syscalls) supported: `read()`, `write()`, ...

/sys

- Folders in each sysfs directory correspond to objects (kobjects)
- Files in sysfs represent attributes of an object
  - write file → change attribute value
  - read file → retrieve attribute value

Generic data structures

- The Linux kernel implements various generic data structures
  - Doubly linked lists
  - Queues
  - Maps
  - Binary trees
  - ...
- Avoid dynamic memory allocation when possible



Bottom-half methods

Motivation

- Sometimes, a certain kind of work **cannot be done immediately**
  - Too much computation associated with interrupt processing
    - TCP/IP processing upon recieving network packet
  - Inability to invoke blocking functions from interrupt context and other code regions

Solution: defer work for later

- Sofirqs → `raise_sofirq()`
- Tasklets → `tasklet_schedule()`
- Work Queues → `schedule_work()`

Mechanisms to defer work

- Deferred work (task) is modeled as a structure with an associated function
- The task is first **scheduled** and will be **executed** later (with interrupts enabled)

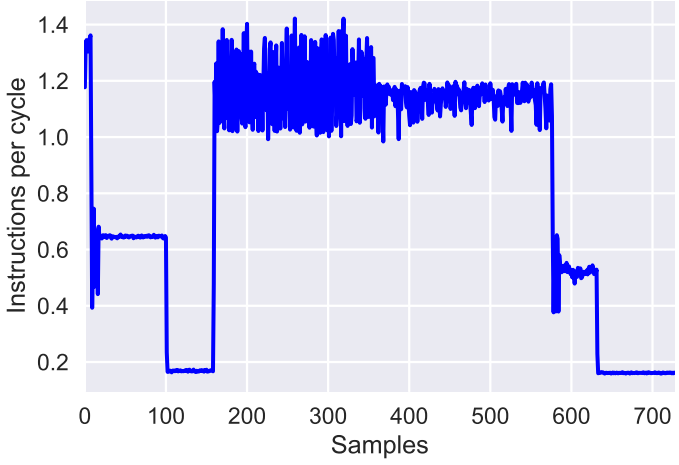
Mechanism	Advantages/Disadvantages	Work descriptor	Context
Softirqs	High Performance Requires changes to the kernel Concurrency issues	`softirq_action` `<linux/interrupt.h>`	Interrupt Context[1]
Tasklets	Softirqs from kernel modules Concurrency easy to handle Limited scalability	`tasklet_struct` `<linux/interrupt.h>`	Interrupt Context[1]
Workqueues	Run in process context Easy to use Lower performance	`work_struct` `<linux/workqueue.h>`	Process Context

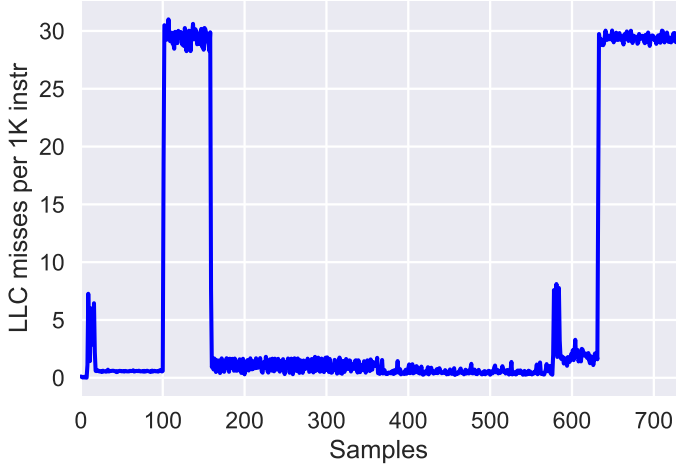
[1] When system is loaded with sofirq processing, the `ksoftirqd` kernel threads executes softirqs.

Outline

Performance monitoring counters

- Most modern complex computing systems are equipped with hardware Performance Monitoring Counters (PMCs)
- High-level performance metrics collected via PMCs provide valuable hints to programmers and computer architects
  - IPC, Last-Level Cache (LLC) miss rate, ...





Performance monitoring counters

PMCs: Registers accessible via special ISA instructions

- Two types:
  1. Fixed-function PMC
  2. General-purpose (configurable) PMC
- Each PMC has a control register associated to it
  - Enable, Event Type, Subevent (UMASK), Interrupt-related behavior, ...
- Direct access to PMCs is typically restricted to code running at the OS privilege level
  - Kernel-level tools enable users to access PMCs
  - Low-level access to PMCs is tedious

PMCs and the OS scheduler

- *The OS scheduler can leverage PMCs to perform effective optimizations in modern multicore systems (CMPs)*
  - Overall Idea:
    1. OS characterizes application behavior online using PMCs
    2. Perform thread-to-core mappings to optimize a certain metric

Main challenge:

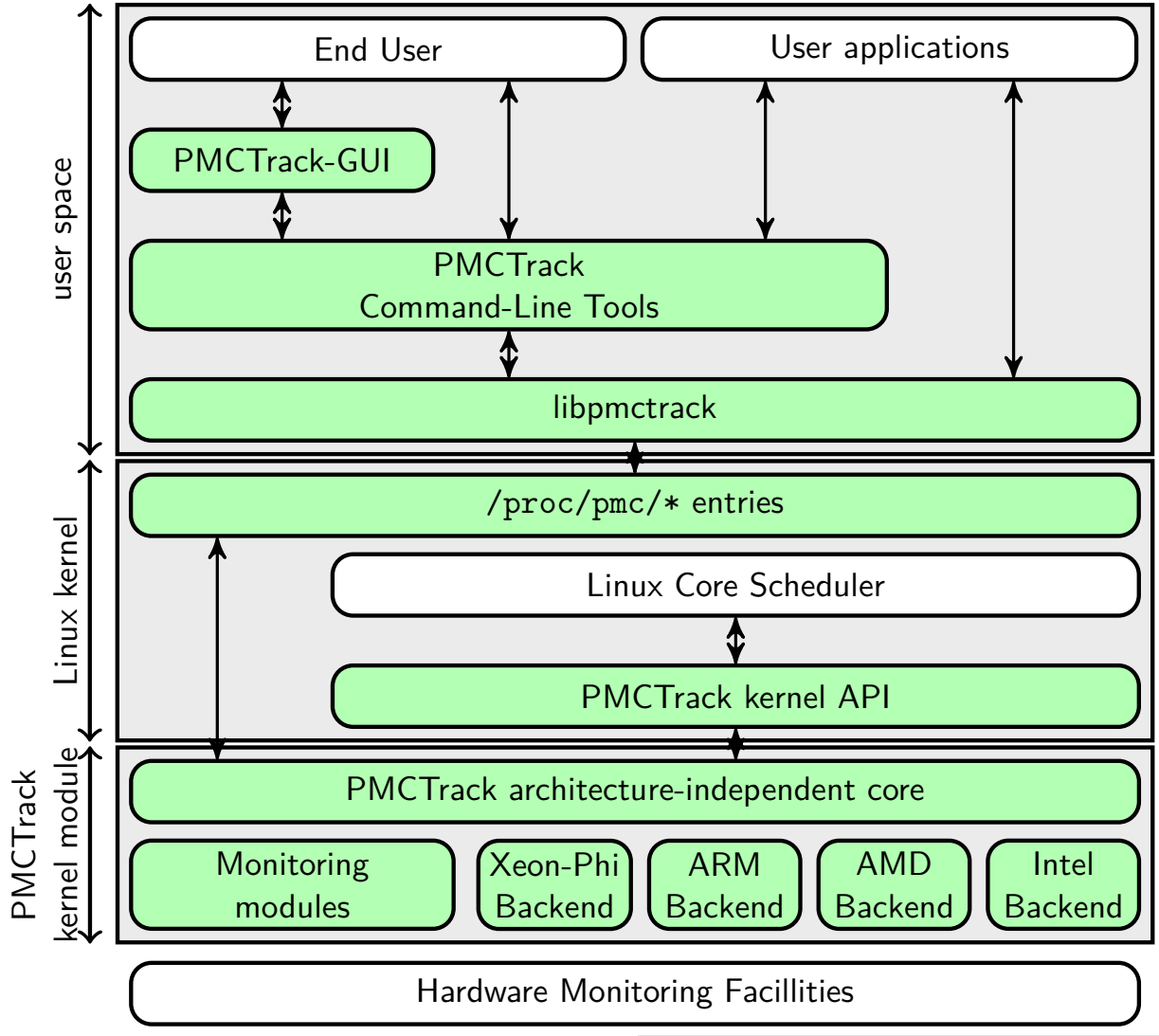
How to use HW counters within the Linux scheduler without polluting the implementation with platform-specific code

The PMCTrack performance monitoring tool

PMCTrack

- Project started in 2007
  - It provided access to PMCs from the scheduler code only
- Contributions made by UCM students
  - 2012: *Guillermo Martínez, Sergio Sánchez, Sofía Dronda*
  - 2015: *Jorge Casas, Abel Serrano*
  - 2016: *Adrián García, Álvaro Sanz*
  - ...
- Today, it is an open-source tool for the Linux kernel (GPL v2)
  - Supports both user-space and kernel-level monitoring
  - Other monitoring information beyond HW PMC events:
    - Energy/Power consumption readings (Intel/ARM)
    - Last-level cache usage (Intel CMT)

PMCTrack architecture



Using PMCTrack from user space

Usage modes

1. **Time-Based Sampling (TBS)**
   - An application's PMC and virtual counter values are collected at regular time intervals
2. **Time-Based system-wide monitoring mode**
   - TBS for each CPU in the system
3. **Event-Based Sampling (EBS)**
   - An application's PMC and virtual counter values are collected when a given HW event counter reaches a certain count
4. **Self-monitoring mode** (instrumentation with *libpmctrack*)
   - Retrieve PMC and virtual counter values for specific code fragments

The pmctrack command-line tool (TBS)

TBS with pmctrack

```
$ pmctrack  -T 1  -c instr,cycles,llc_misses -V energy_core ./mcf06
[Event-to-counter mappings]
pmc0=instr
pmc1=cycles
pmc3=llc_misses
virt0=energy_core
[Event counts]
```

nsample	pid	event	pmc0	pmc1	pmc3	virt0
1	7778	tick	1989870049	3521428186	30089533	7869018
2	7778	tick	1234494786	3587014459	25088834	7564880
3	7778	tick	1218749172	3586903894	24850423	7397277
4	7778	tick	1554203899	3271717037	19625102	8395812
5	7778	tick	1651034822	3025477428	10236088	8309570
6	7778	tick	2622803674	3586074561	18181359	6571350
7	7778	tick	2144785721	3591965814	19974329	6329467
8	7778	tick	1362794547	3591687395	22918211	5958740
9	7778	tick	1449585203	3591073740	22429638	6103515
10	7778	tick	1334665386	3590503009	22791454	5949584
11	7778	tick	1343737278	3591222149	22656358	5937927
12	7778	tick	1354545497	3589517197	22481647	6004760
13	7778	tick	1384381286	3592969591	22269118	5994689
14	7778	tick	1322495773	3592560690	22350731	6013854

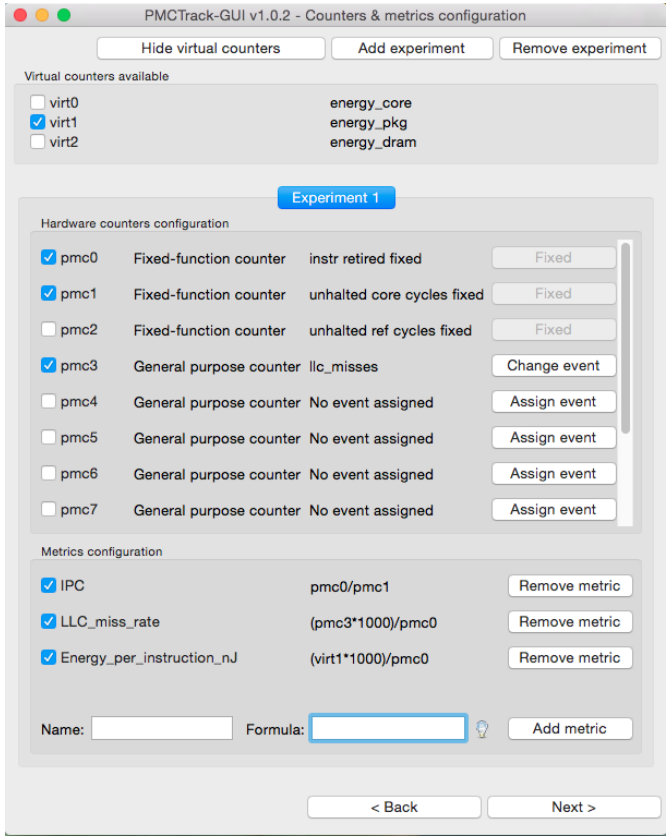
```
...
```

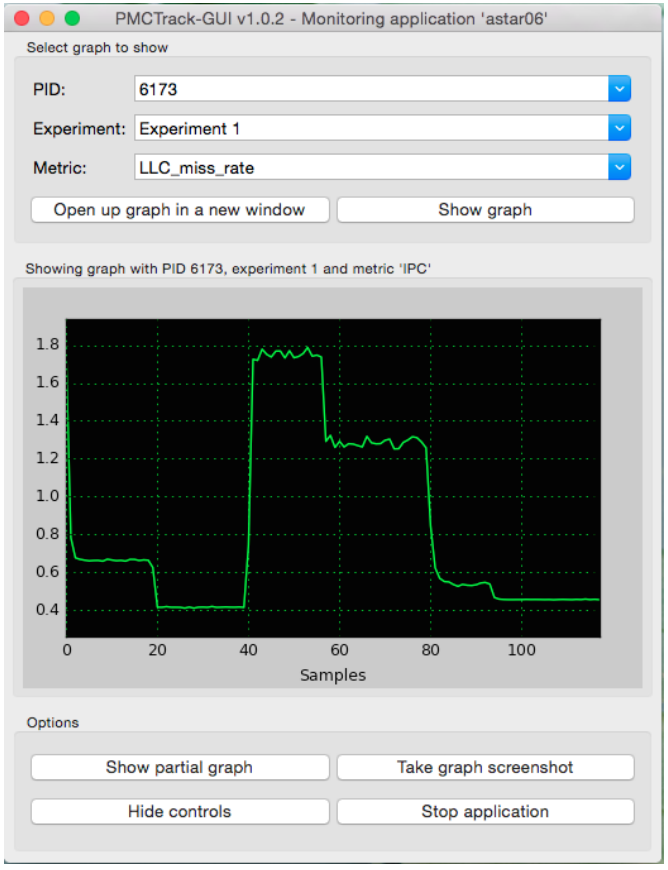
The pmctrack command-line tool (TBS)

TBS with pmctrack + pmc-metric

```
$ pmctrack  -T 1  -c instr,cycles,llc_misses -V energy_core ./mcf06 \
    | pmc-metric -v -m 'IPC=pmc0/pmc1' -m 'LLCMPKI=(1000*pmc3/pmc0)' -m 'EPI=(1000*virt0)/pmc0'
[Event-to-counter mappings]
pmc0=instr
pmc1=cycles
pmc3=llc_misses
virt0=energy_core
....
nsample    pid      event          IPC      LLCMPKI          EPI
      1   7843       tick     0.561232    15.257263     4.006190
      2   7843       tick     0.342564    20.487803     6.206797
      3   7843       tick     0.336885    20.794879     6.299878
      4   7843       tick     0.467948    13.102071     5.444737
      5   7843       tick     0.550340     6.039147     5.127041
      6   7843       tick     0.726642     7.018734     2.470913
      7   7843       tick     0.602763     9.199328     2.960916
      8   7843       tick     0.379303    16.821609     4.276329
      9   7843       tick     0.405407    15.388447     4.163115
     10   7843       tick     0.369597    17.169366     4.385655
     11   7843       tick     0.376348    16.728609     4.448713
     12   7843       tick     0.377036    16.614485     4.339088
     13   7843       tick     0.384078    16.130389     4.323180
     14   7843       tick     0.370209    16.777249     4.443311
     15   7843       tick     0.376361    16.289599     4.430110
     16   7843       tick     0.371755    16.375152     4.486414
     17   7843       tick     0.379319    15.409991     4.542519
...
```

PMCTrack-GUI: a Python graphical frontend





PMCTrack open-source project

- http://pmctrack.dacya.ucm.es
- https://github.com/jcsaezal/pmctrack

Future Work

- Support for MIPS Architecture
- Energy/Power readings with Odroid Smart Power 2
- ...

Conclusions

- The Linux kernel is becoming increasingly complex
  - The documentation becomes outdated quickly
  - Developers must be capable of understanding the kernel code
- Basic knowledge of main kernel abstractions is required
  - Pseudo FSs, system calls, kernel timers, BHs, ...
- Linux kernel modules make it easier to get started
- If you're interested, enroll in "Linux and Android Internals"
  - *Arquitectura Interna de Linux y Android*

Questions