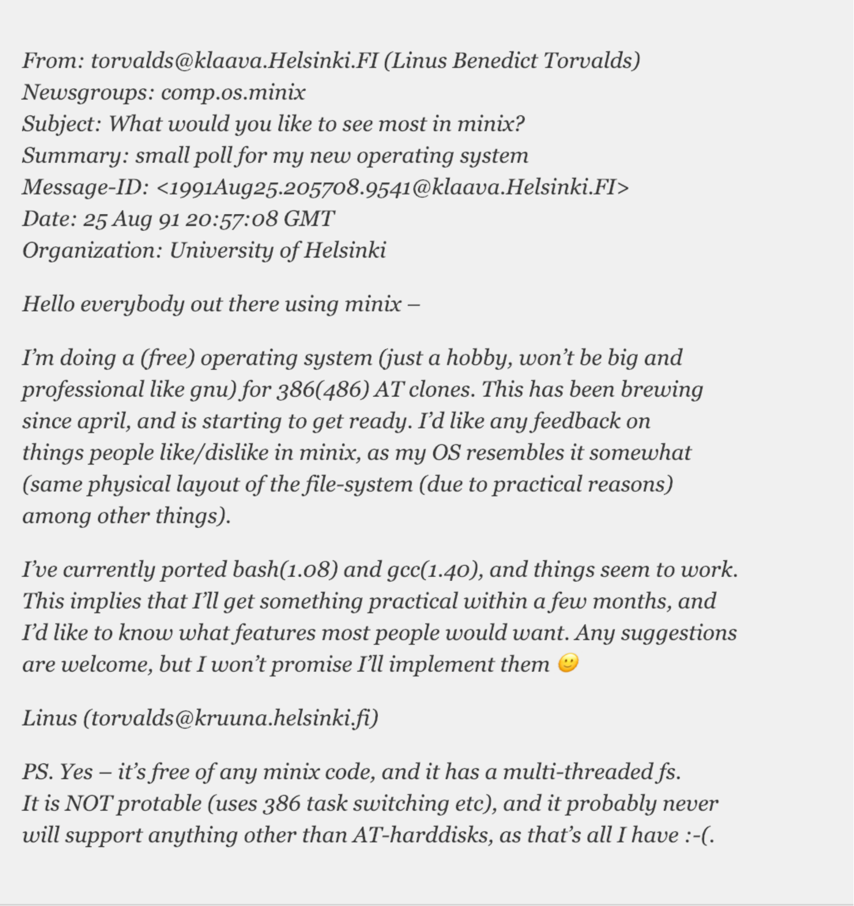
Operating Systems and C
7a. Linux Kernel Security

With slides from Hans Holmberg

Outline

- Context
- Kernel
- Community
- Loadable Modules
- Boot Process
- Key Concepts
- Linux Security Frameworks

Linux First Announcement



From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

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 (torvalds@kruuna.helsinki.fi)

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

Why this course?

https://github.com/torvalds/linux

https://gcc.gnu.org/

POSIX

Portable Operating System Interface

Goal: Common denominator for Unix systems

Collection of Specifications: Core services (processes, signals, File system, Pipes, **I/O**, C Library), Real-time extensions, **Threads**.

GNU

"GNU, which stands for **Gnu's Not Unix**, is the name for the complete Unix-compatible software system which I am writing so that I can give it away free to everyone who can use it. Several other volunteers are helping me. Contributions of time, money, programs and equipment are greatly needed."

GNU Manifesto

Richard Stallman, 1985

GNU/Linux

By 1991, the GNU ecosystem contained:

- A C compiler: gcc (1st version in 1987)
- A standard C library: glibc
- A text editor: Emacs

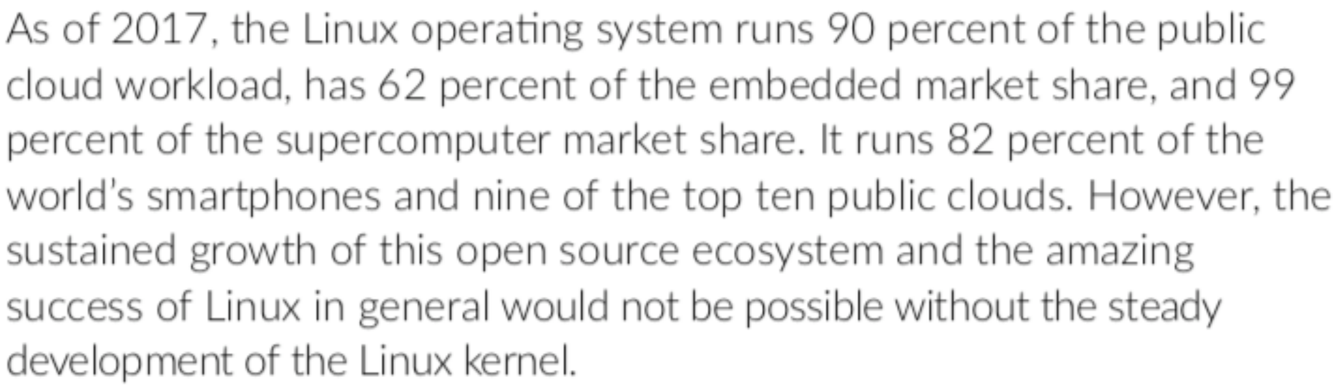
No full kernel implementation => Linux fixed that.

Linux

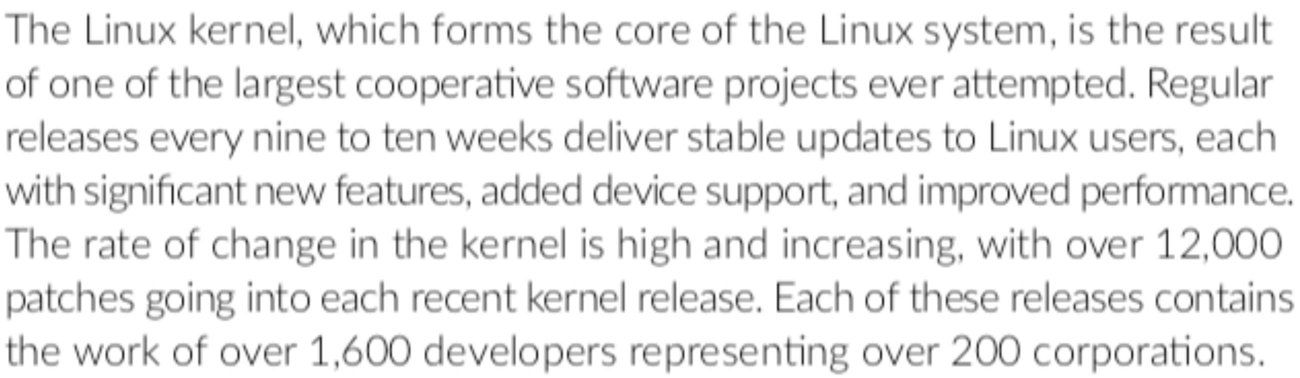
- Linux is a Registered Trademark of Linus Torvalds.
- Mostly POSIX-compliant OS:
  - Kernel: Monolithic OS kernel
  - Linux Distribution: Kernel, GNU tools and libraries, package management system, documentation, window system, window manager, desktop environment
    - E.g., Ubuntu, Red Hat, Gentoo, Arch Linux, Mint, …
  - Android : Mobile OS
  - Yocto: Templates, tools and methods to help you create custom Linux-based systems for embedded and IOT products

Linux Today

https://www.linuxfoundation.org/2017-linux-kernel-report-landing-page/
https://www.linuxcounter.net/



As of 2017, the Linux operating system runs 90 percent of the public cloud workload, has 62 percent of the embedded market share, and 99 percent of the supercomputer market share. It runs 82 percent of the world's smartphones and nine of the top ten public clouds. However, the sustained growth of this open source ecosystem and the amazing success of Linux in general would not be possible without the steady development of the Linux kernel.

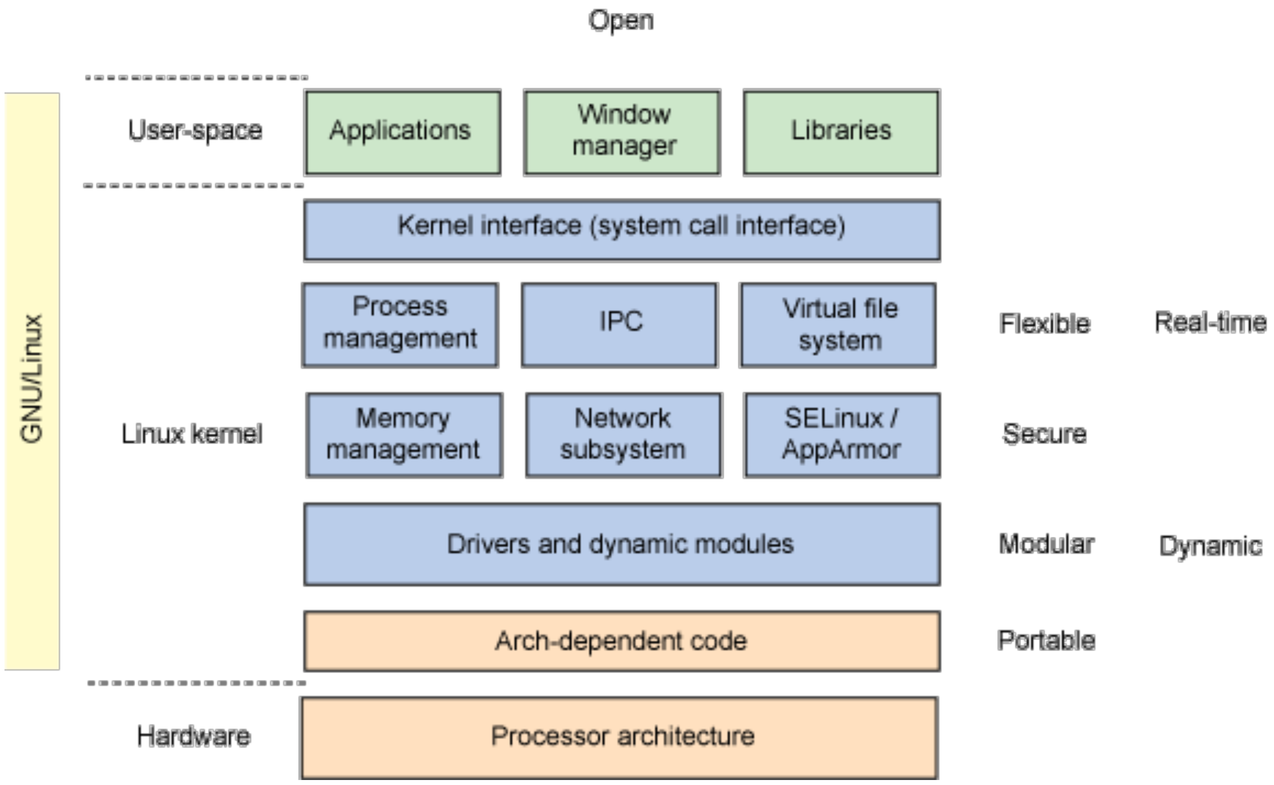


The Linux kernel, which forms the core of the Linux system, is the result of one of the largest cooperative software projects ever attempted. Regular releases every nine to ten weeks deliver stable updates to Linux users, each with significant new features, added device support, and improved performance. The rate of change in the kernel is high and increasing, with over 12,000 patches going into each recent kernel release. Each of these releases contains the work of over 1,600 developers representing over 200 corporations.

Outline

- Context
- Kernel
- Community
- Loadable Modules
- Boot Process
- Key Concepts
- Linux Security Frameworks

Architecture



CPU Modes

- 4 protection rings in X86_64
  - Instructions at Ring X, not available from Ring X+1
- Ring 0 is most privileged
  - Accessible from **Linux kernel**
- Ring 3 is least priviledged
  - Accessible from Linux user space

Example privileged instructions:

HLT: Halt CPU till next interrupt.

INVLPG: Invalidate a page entry in the translation look-aside buffer (TLB).

LIDT: Load Interrupt Descriptor Table.

MOV CR registers: load or store control registers. In this case the MOV instruction (a non-privileged instruction on its own) is accessing a privileged register.

Modify IO privilege level

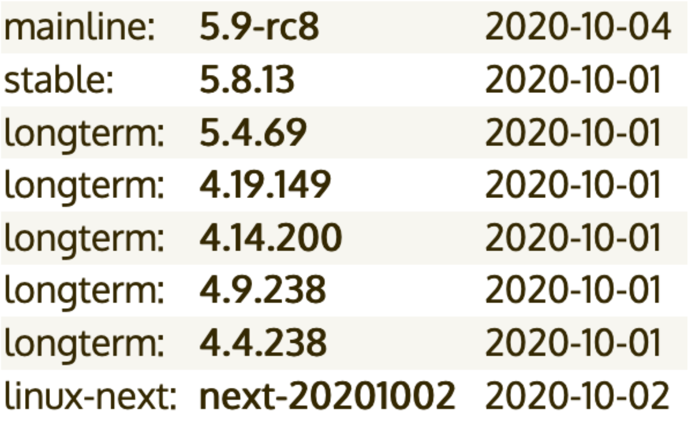
OS Kernel

- The OS kernel is started when the computer boots
- The OS kernel then manages all the computer’s resources (processor, memory, I/O devices)
- The OS kernel partitions the memory into kernel space (reserved to the kernel) and user space (all applications)
- The OS kernel exposes an interface to user space applications, the system calls.

Kernel

- Debugging is hard
  - Bugs bring the system down!
- No standard library (no libc, no headers)
  - No libc support for threads, I/Os, data structures.
  - Kernel-specific services
- No memory protection mechanism
- No high-level abstraction for floating points
- Small per-process fixed stack
- Preemptive tasks, asynchronous interrupts, supports for multi-processing (SMP)
  - Synchronization and concurrency are hard to manage!
- Portability is of the essence
  - Avoid undefined behavior!
  - Endian-neutral, no assumptions about page/word size, …

Kernel Source Code



mainline:	**5.9-rc8**	2020-10-04
stable:	**5.8.13**	2020-10-01
longterm:	**5.4.69**	2020-10-01
longterm:	**4.19.149**	2020-10-01
longterm:	**4.14.200**	2020-10-01
longterm:	**4.9.238**	2020-10-01
longterm:	**4.4.238**	2020-10-01
linux-next:	**next-20201002**	2020-10-02

- Available from kernel.org
- Several versions of the kernel:
  - Mainline (e.g., 5.9-rc8)
    - Maintained by Linus Torvald, benevolent dictator
    - Master tree, all new code is merged here
  - Stable and longterm
    - Maintained by Greg Kroah-Hartman and others
    - Bug fixes and trivial support for new devices
  - Next
    - Maintained by Stephen Rothwell
    - Staging ground for new code from the maintainers

Outline

- Context
- Kernel
- Community
- Loadable Modules
- Boot Process
- Key Concepts
- Linux Security Frameworks

Linux Kernel Community

Julia Lawall (INRIA, ex DIKU) - Jens Axboe (FB)
Coccinelle Block Layer (fio)

https://github.com/netoptimizer
Jesper Dangaard Brouer

Linux Kernel Community

Lectures

Theory

- Linux Kernel overview
- Kernel boot and machine descriptions (ACPI/DT)
- Interrupts, threads and processes
- Power management

Craft

- Navigating the source tree
- Configuring and building the kernel
- Debugging and tracing

Community

- Development best practices
- Community interaction and upstreaming

Hans Holmberg (WD)
Kernelteaching
https://lundlinuxcon.org/

LightNVM: The Linux Open-Channel SSD Subsystem
IT UNIVERSITY OF COPENHAGEN
CNEX LABS

Matias Bjørling (WD)
LightNVM

(Samsung)
pblk

Patch-Based Evolution

https://www.kernel.org/doc/html/v4.17/process/submitting-patches.html
https://www.kernel.org/doc/html/v4.17/process/2.Process.html#the-big-picture

Linux is under the responsibility of Linus Thorvald.

Linux is decomposed into subsystems, under the responsibility of a maintainer (e.g., Jens Axboe for the block layer)

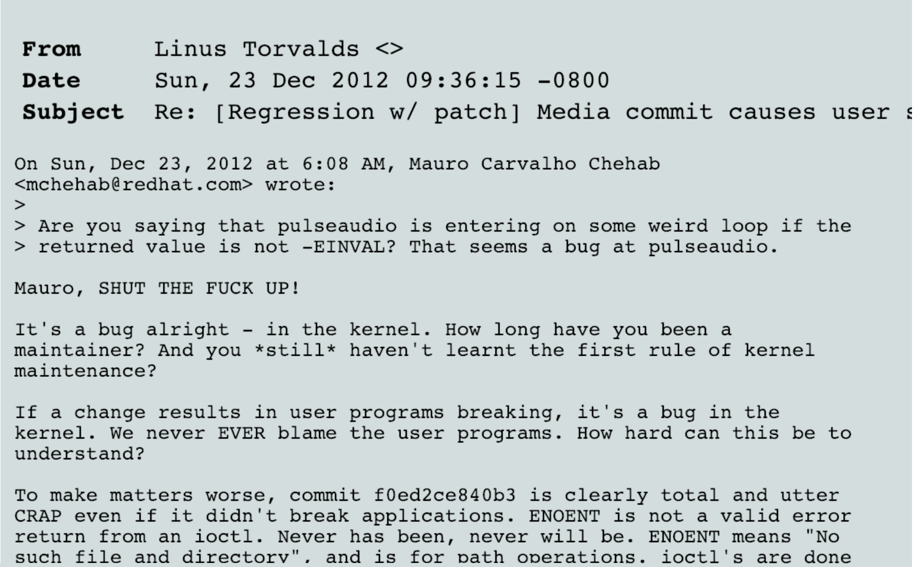
- Each maintainer is a gatekeeper for her subsystem
- They manage their version of the source tree
- Review/accept patches from developers
- Send pull requests to Linus for patches that they think should be merged into the mainline

Submitting a Patch

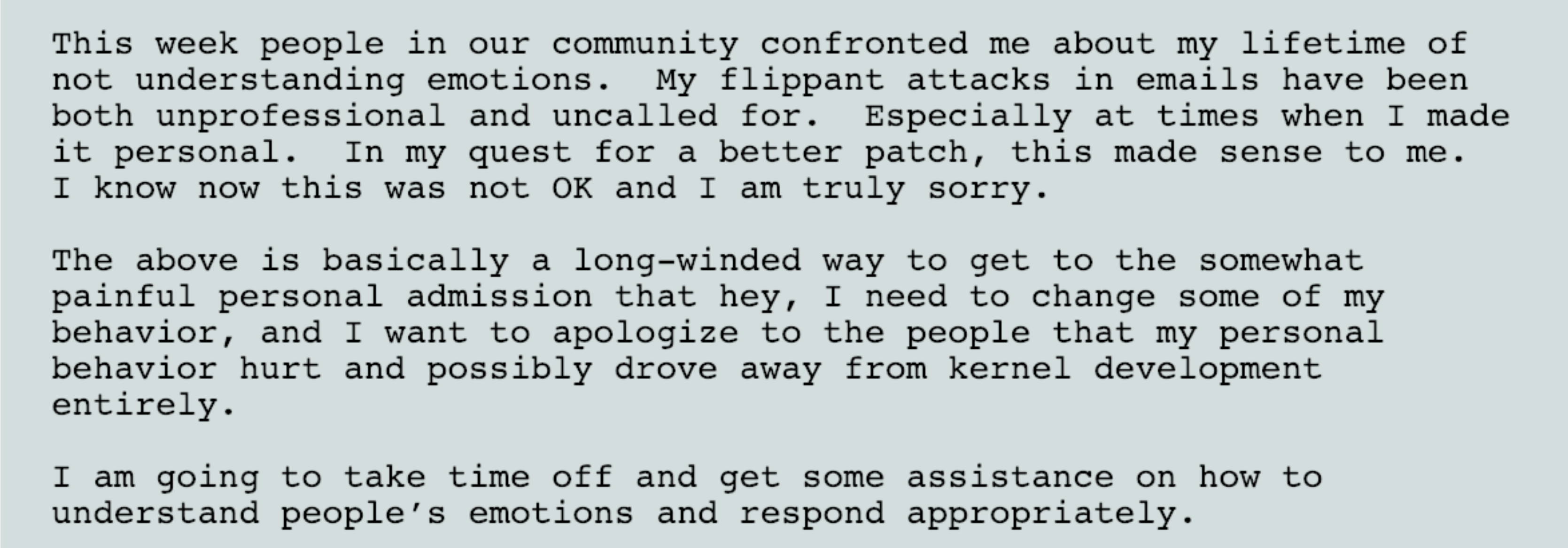
1. Git as a tool to represent diff
2. Describe changes
3. Make sure your code conforms to Linux coding style
4. Send patch to relevant reviewer (plain text)
5. Respond to comment from reviewer
6. Reviewer signs off your patch and forwards to maintainer

Code of Conflict

[Linux Code of conduct]



```
From      Linus Torvalds <>
Date      Sun, 23 Dec 2012 09:36:15 -0800
Subject   Re: [Regression w/ patch] Media commit causes user s

On Sun, Dec 23, 2012 at 6:08 AM, Mauro Carvalho Chehab
<mchehab@redhat.com> wrote:
>
> Are you saying that pulseaudio is entering on some weird loop if the
> returned value is not -EINVAL? That seems a bug at pulseaudio.

Mauro, SHUT THE FUCK UP!

It's a bug alright - in the kernel. How long have you been a
maintainer? And you *still* haven't learnt the first rule of kernel
maintenance?

If a change results in user programs breaking, it's a bug in the
kernel. We never EVER blame the user programs. How hard can this be to
understand?

To make matters worse, commit f0ed2ce840b3 is clearly total and utter
CRAP even if it didn't break applications. ENOENT is not a valid error
return from an ioctl. Never has been, never will be. ENOENT means "No
such file and directory", and is for path operations. ioctl's are done
                                                   way in hell that

                                                   regression,
                                                   has some serious

                                                   t kind of obvious
                                                   Seriously.

                                                   I have another
                                                   tions being broken
                                                   d you've shown
                                                    apply it directly

                                                   particularly don't
                                                   e your whole email
                                                   hings was so
                                                   n shit. It adds an
                                                   o insane, it adds a
                                                   : ret").

                                                   eaking user space,
                                                   rk, is just

Fix your f*cking "compliance tool", because it is obviously broken.
And fix your approach to kernel programming.

               Linus
```



This week people in our community confronted me about my lifetime of not understanding emotions. My flippant attacks in emails have been both unprofessional and uncalled for. Especially at times when I made it personal. In my quest for a better patch, this made sense to me. I know now this was not OK and I am truly sorry.

The above is basically a long-winded way to get to the somewhat painful personal admission that hey, I need to change some of my behavior, and I want to apologize to the people that my personal behavior hurt and possibly drove away from kernel development entirely.

I am going to take time off and get some assistance on how to understand people’s emotions and respond appropriately.

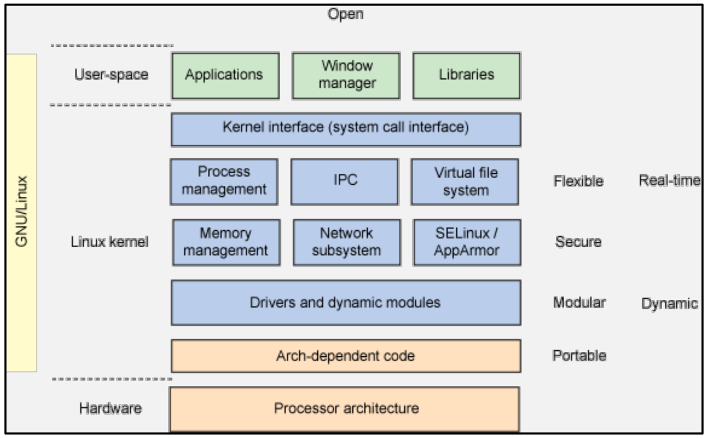
Outline

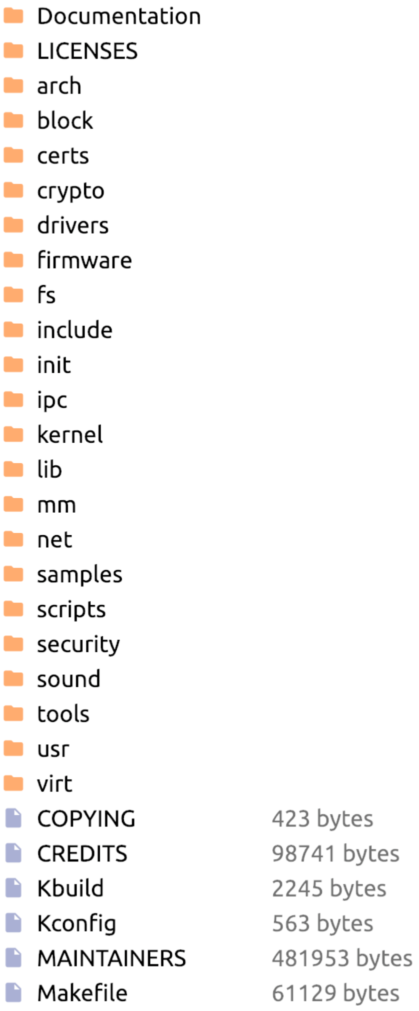
- Context
- Kernel
- Community
- Loadable Modules
- Boot Process
- Key Concepts
- Linux Security Frameworks

Kernel Source Code

https://elixir.bootlin.com/linux/latest/source

- **kernel**: core kernel code
- **arch**: architecture specific
- **mm**: Memory Management
- **net**: Network stacks
- **fs:** File Systems
- **block**: Block Layer
- **drivers**: device drivers and loadable modules
- **documentation**
- **scripts**: utilities



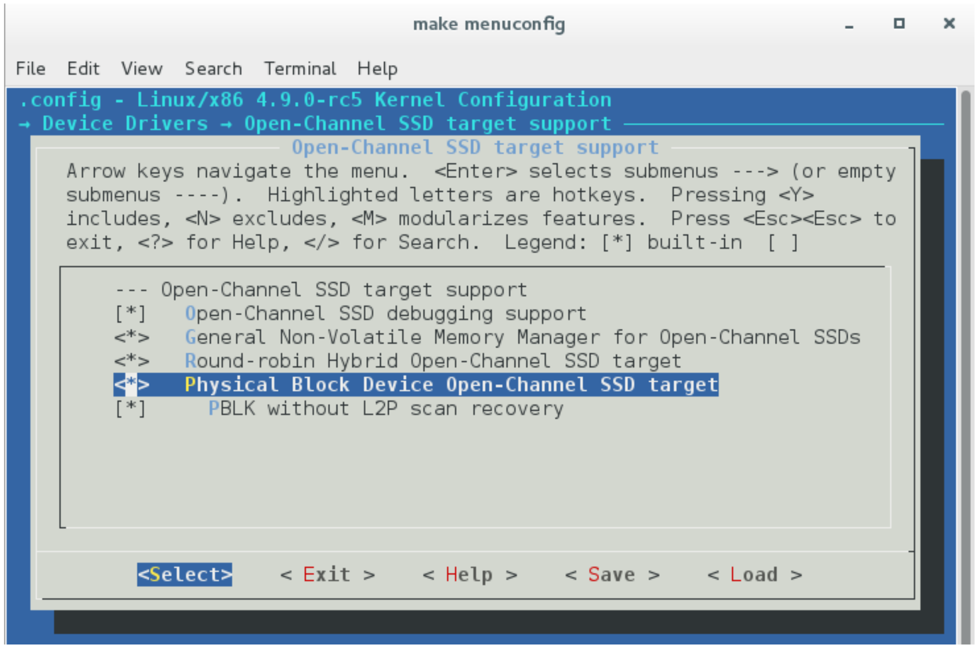


🗀 Documentation	
🗀 LICENSES	
🗀 arch	
🗀 block	
🗀 certs	
🗀 crypto	
🗀 drivers	
🗀 firmware	
🗀 fs	
🗀 include	
🗀 init	
🗀 ipc	
🗀 kernel	
🗀 lib	
🗀 mm	
🗀 net	
🗀 samples	
🗀 scripts	
🗀 security	
🗀 sound	
🗀 tools	
🗀 usr	
🗀 virt	
🗎 COPYING	423 bytes
🗎 CREDITS	98741 bytes
🗎 Kbuild	2245 bytes
🗎 Kconfig	563 bytes
🗎 MAINTAINERS	481953 bytes
🗎 Makefile	61129 bytes

Building the kernel

Configuring: .config file or make menuconfig

- Configuration options (about HW, features)
- Which drivers to build (about peripherals)
- Debug options



```
.config - Linux/x86 4.9.0-rc5 Kernel Configuration
→ Device Drivers → Open-Channel SSD target support
                    Open-Channel SSD target support
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
  submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
  exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]

      --- Open-Channel SSD target support
      [*]   Open-Channel SSD debugging support
      <*>   General Non-Volatile Memory Manager for Open-Channel SSDs
      <*>   Round-robin Hybrid Open-Channel SSD target
      <*>   Physical Block Device Open-Channel SSD target
      [*]     PBLK without L2P scan recovery

        <Select>    < Exit >    < Help >    < Save >    < Load >
```

Building the kernel

Kernel files generated by the build process, placed under /boot:

- Linux kernel executable:
  - vmlinux, vmlinuz: vm for virtual memory, z for compressed
- Linux kernel image (that can be loaded as is in RAM so that it can be executed):
  - zImage, bzImage, uImage
- Initial RAMDisk
  - Initial root file system
  - Enough drivers so that the kernel can mount/start initializing
- Device Tree Structure (.dtbs)
  - Depending on Processor/System devices
- Loadable Kernel Modules (.ko)
  - Built at compile time; enabled at *boot time*; loaded at run-time
- System Map
  - Map (Symbol table, Address in memory)

Devices Drivers

A device driver:

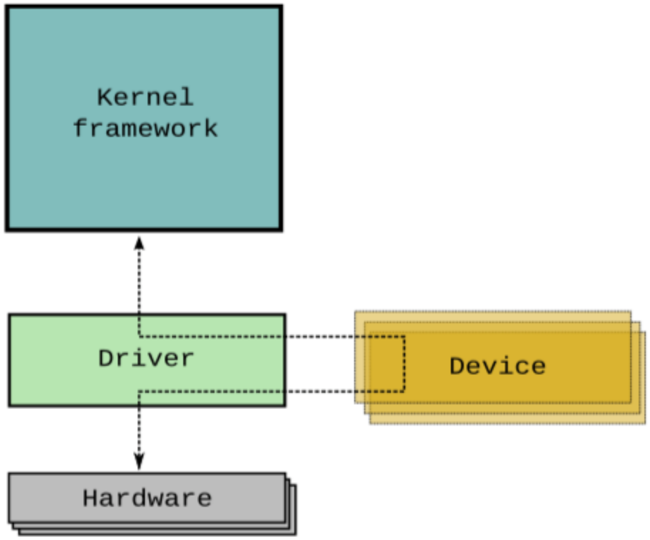
- enables the operating system to interact with a piece of hardware.
- aims to abstract the hardware specific properties away
- provides access to it via an interface shared with other devices to a common kernel framework(i.e. input, iio, ..)

Devices and Drivers

3 classes of devices:

1. Block Devices
   - Block I/O and Virtual File System (upcoming lecture)
2. Character Devices
3. Network Devices
   - Accessed via socket API (breaks everything is a file)

Drivers provide code handling a class of hardware, device objects contain the specific state for a single piece of hardware.



Driver Lifecycle

- Init - global initialization
- Probe - create device
- Open/Close
- Power management
- Remove - global deinitialization

Builtin Drivers vs. Loadable Modules

http://www.tldp.org/HOWTO/html_single/Module-HOWTO

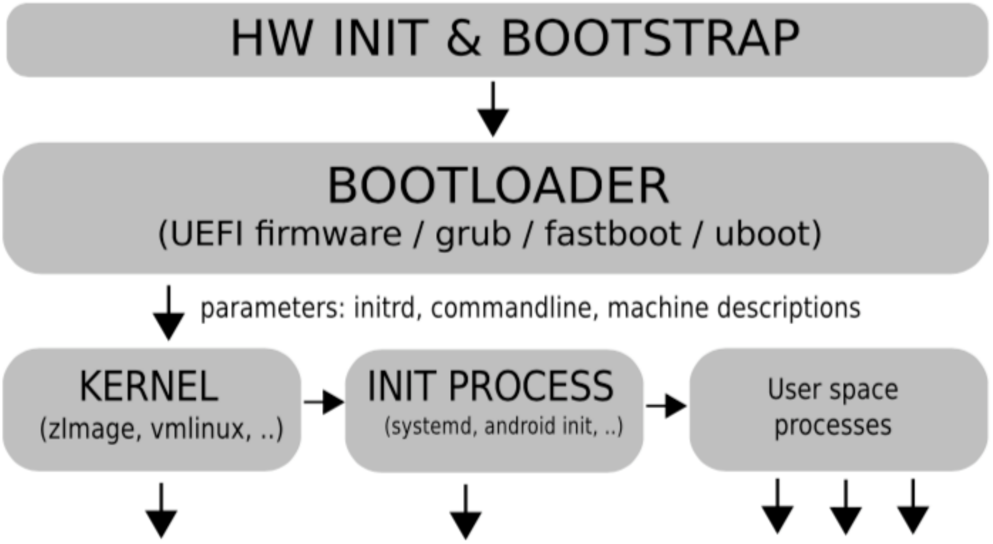
- Code that does not need to be executed before a filesystem is available to the kernel can be compiled as a kernel module.
- Kernel modules saves a lot of memory!
- Modules can be loaded automatically, if the module provides a device table that can be matched with well-known device ids, e.g., usb device ids.

Outline

- Context
- Kernel
- Community
- Loadable Modules
- Boot Process
- Key Concepts
- Linux Security Frameworks

Linux Boot Process

Kernel boot process overview



HW INIT & BOOTSTRAP

BOOTLOADER
(UEFI firmware / grub / fastboot / uboot)

parameters: initrd, commandline, machine descriptions

KERNEL
(zImage, vmlinux, ..)

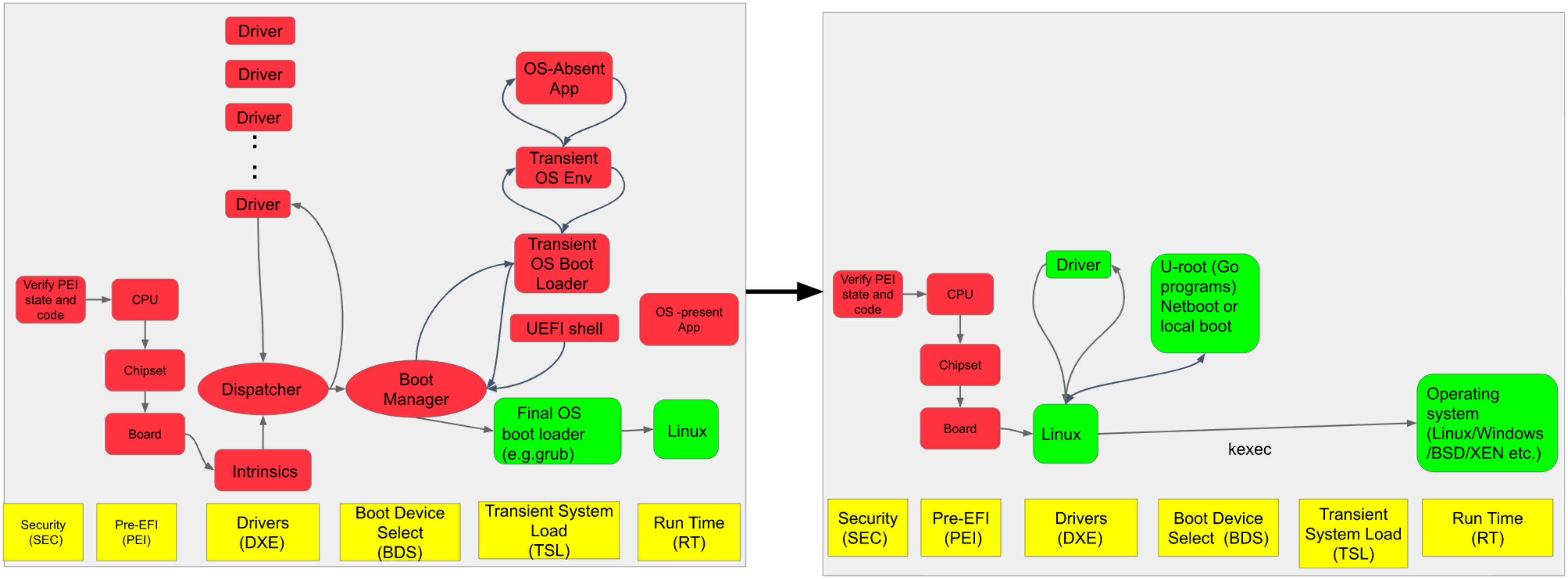
INIT PROCESS
(systemd, android init, ..)

User space processes

Hardware and firmware vulnerabilities

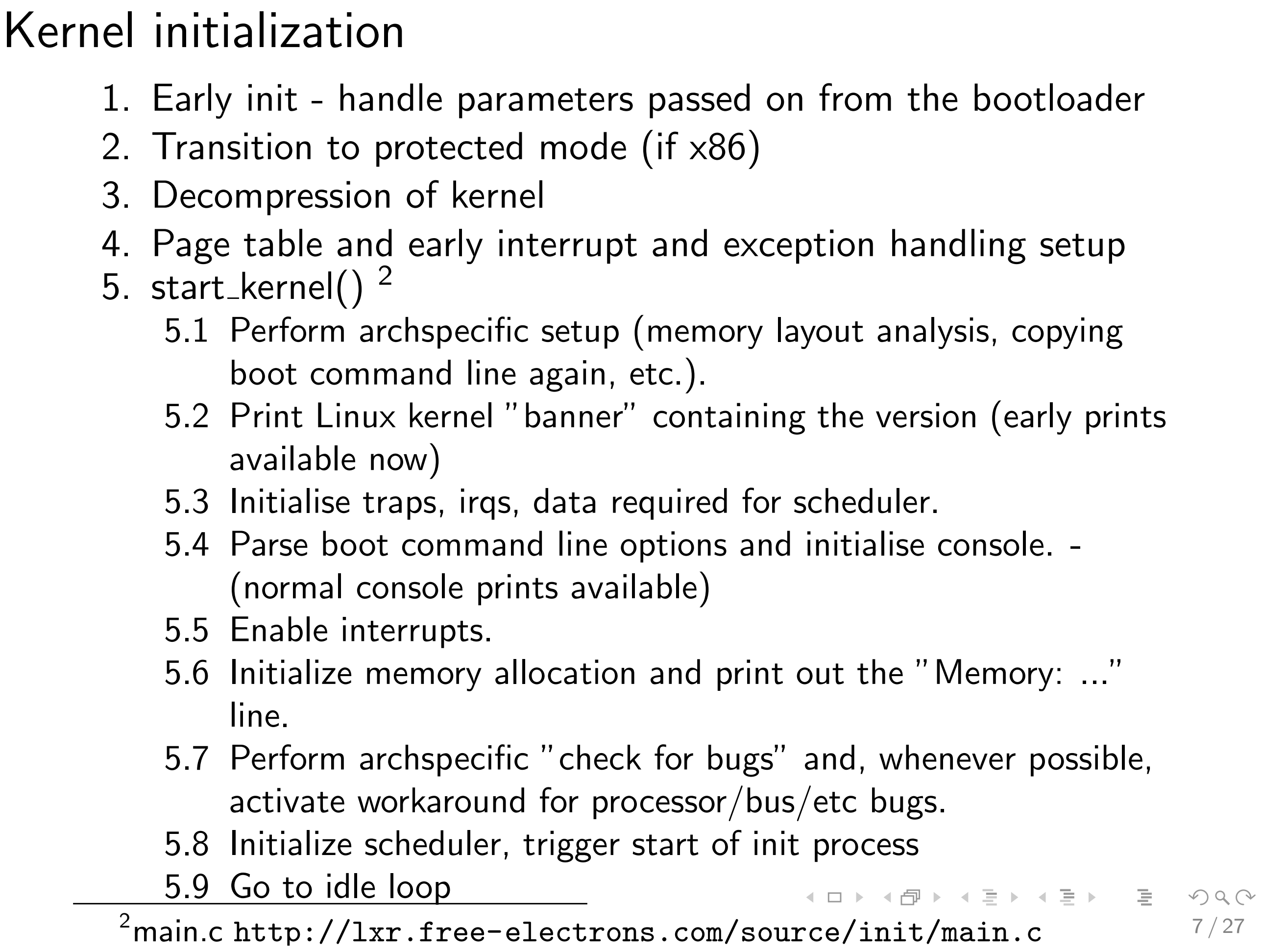
LinuxBoot

Linux as firmware: LinuxBoot



https://www.opencompute.org/projects/open-system-firmware

Linux Boot Process



Kernel initialization

1. Early init - handle parameters passed on from the bootloader
2. Transition to protected mode (if x86)
3. Decompression of kernel
4. Page table and early interrupt and exception handling setup
5. start_kernel() [2]
   - 5.1 Perform archspecific setup (memory layout analysis, copying boot command line again, etc.).
   - 5.2 Print Linux kernel "banner" containing the version (early prints available now)
   - 5.3 Initialise traps, irqs, data required for scheduler.
   - 5.4 Parse boot command line options and initialise console. - (normal console prints available)
   - 5.5 Enable interrupts.
   - 5.6 Initialize memory allocation and print out the "Memory: ..." line.
   - 5.7 Perform archspecific "check for bugs" and, whenever possible, activate workaround for processor/bus/etc bugs.
   - 5.8 Initialize scheduler, trigger start of init process
   - 5.9 Go to idle loop

[2] main.c `http://lxr.free-electrons.com/source/init/main.c`

Outline

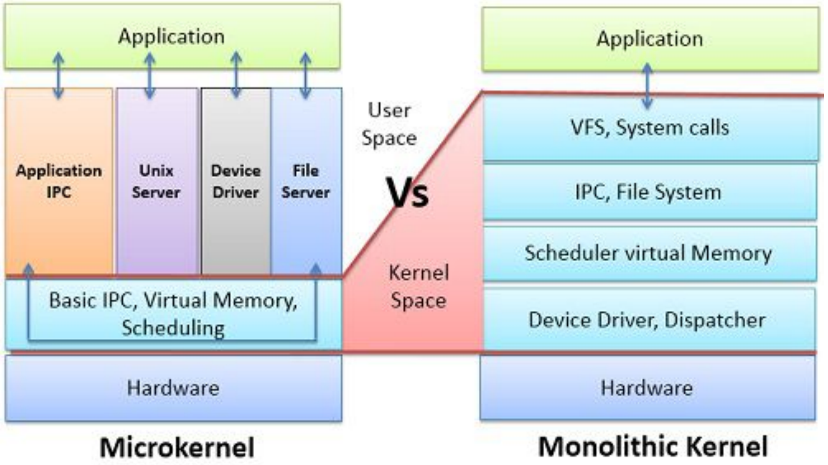
- Context
- Kernel
- Community
- Loadable Modules
- Boot Process
- Key Concepts
- Linux Security Frameworks

Kernel Key Concepts

- Monolithic Kernel
- Stable ABI
- Dynamic loading of kernel modules
- Threads are processes that share resources with other processes
- Everything is a file descriptor
- In-kernel Virtualization (BPF)

Monolithic Kernel vs. Microkernel

https://en.wikipedia.org/wiki/Tanenbaum%E2%80%93Torvalds_debate



"The real issue, and it's really fundamental, is the issue of sharing address spaces. Nothing else really matters. Everything else ends up flowing from that fundamental question: do you share the address space with the caller or put in slightly different terms: can the callee look at and change the callers state as if it were its own (and the other way around)?"

Linus Torvald, 2006 (in response to A.Tanenbaum article in IEEE Computer, May 2006)

Stable ABI

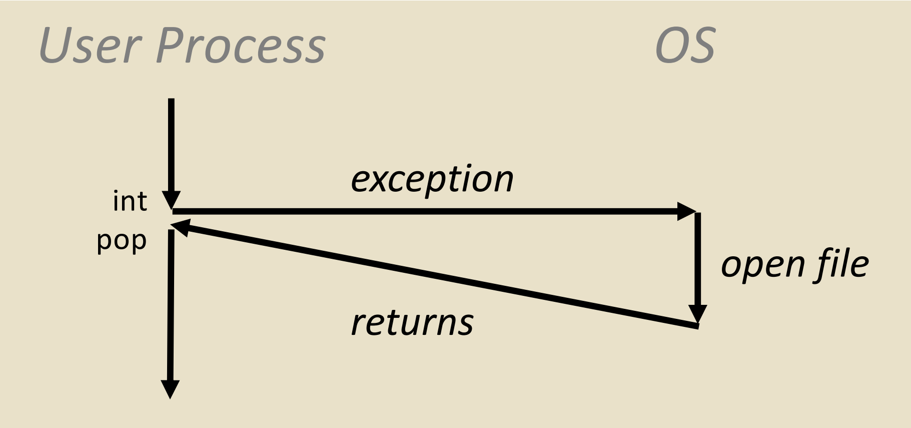
https://github.com/torvalds/linux/tree/master/Documentation/ABI

- ABI: Application Binary Interface
  - API: source code is portable
  - ABI: machine code is portable
- The Linux ABI must be backward compatible and must not break
  - System Call Interface of the Linux kernel
  - Subroutines in the GNU C Library (glibc)

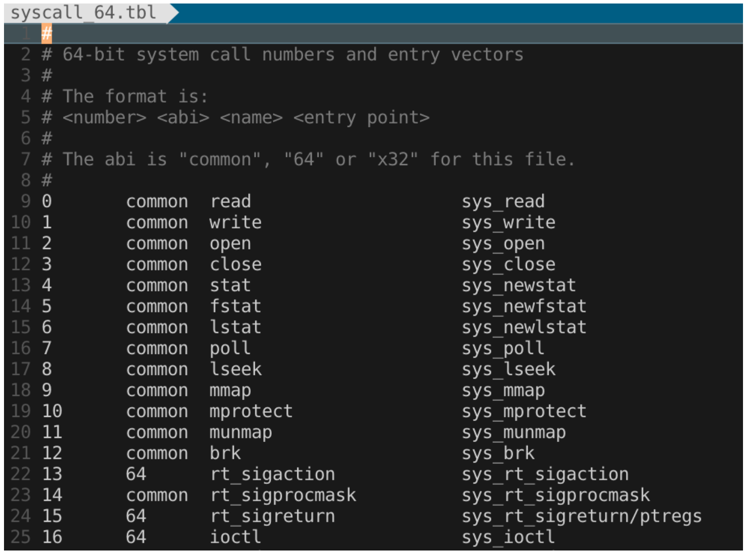
Trap Example: System call

- User calls: `open(filename, options)`
- Function `open` executes **system call instruction `int`**

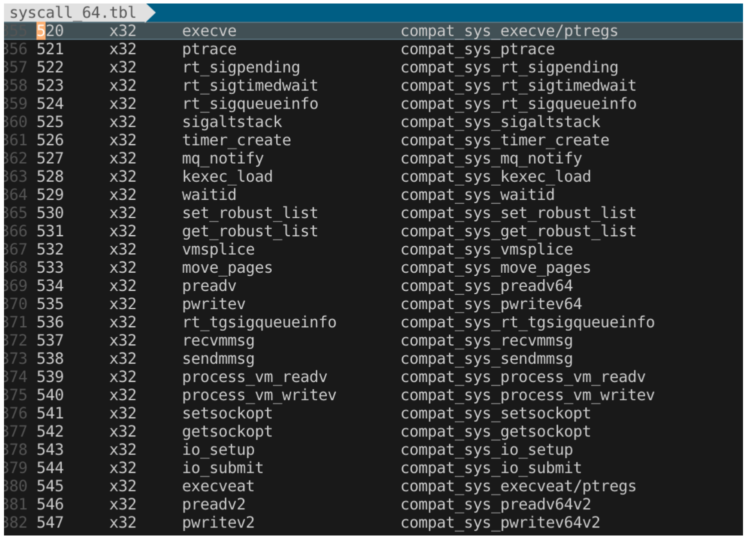
```
0804d070 <__libc_open>:
 . . .
 804d082:    cd 80                  int    $0x80
 804d084:    5b                     pop    %ebx
 . . .
```



- OS must find or create file, get it ready for reading or writing
- Returns integer file descriptor



```
syscall_64.tbl
#
# 64-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The abi is "common", "64" or "x32" for this file.
#
0       common  read                    sys_read
1       common  write                   sys_write
2       common  open                    sys_open
3       common  close                   sys_close
4       common  stat                    sys_newstat
5       common  fstat                   sys_newfstat
6       common  lstat                   sys_newlstat
7       common  poll                    sys_poll
8       common  lseek                   sys_lseek
9       common  mmap                    sys_mmap
10      common  mprotect                sys_mprotect
11      common  munmap                  sys_munmap
12      common  brk                     sys_brk
13      64      rt_sigaction            sys_rt_sigaction
14      common  rt_sigprocmask          sys_rt_sigprocmask
15      64      rt_sigreturn            sys_rt_sigreturn/ptregs
16      64      ioctl                   sys_ioctl
```

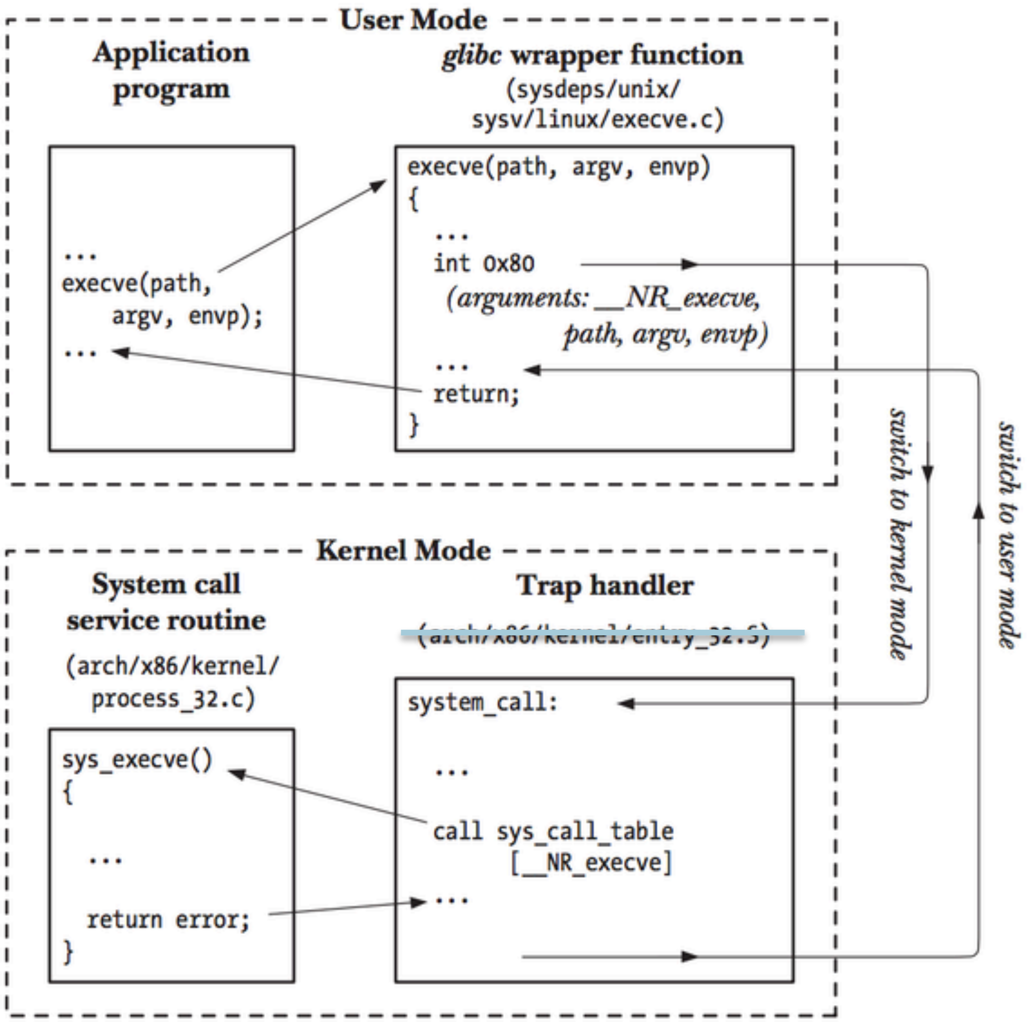


```
syscall_64.tbl
520     x32     execve                  compat_sys_execve/ptregs
521     x32     ptrace                  compat_sys_ptrace
522     x32     rt_sigpending           compat_sys_rt_sigpending
523     x32     rt_sigtimedwait         compat_sys_rt_sigtimedwait
524     x32     rt_sigqueueinfo         compat_sys_rt_sigqueueinfo
525     x32     sigaltstack             compat_sys_sigaltstack
526     x32     timer_create            compat_sys_timer_create
527     x32     mq_notify               compat_sys_mq_notify
528     x32     kexec_load              compat_sys_kexec_load
529     x32     waitid                  compat_sys_waitid
530     x32     set_robust_list         compat_sys_set_robust_list
531     x32     get_robust_list         compat_sys_get_robust_list
532     x32     vmsplice                compat_sys_vmsplice
533     x32     move_pages              compat_sys_move_pages
534     x32     preadv                  compat_sys_preadv64
535     x32     pwritev                 compat_sys_pwritev64
536     x32     rt_tgsigqueueinfo       compat_sys_rt_tgsigqueueinfo
537     x32     recvmmsg                compat_sys_recvmmsg
538     x32     sendmmsg                compat_sys_sendmmsg
539     x32     process_vm_readv        compat_sys_process_vm_readv
540     x32     process_vm_writev       compat_sys_process_vm_writev
541     x32     setsockopt              compat_sys_setsockopt
542     x32     getsockopt              compat_sys_getsockopt
543     x32     io_setup                compat_sys_io_setup
544     x32     io_submit               compat_sys_io_submit
545     x32     execveat                compat_sys_execveat/ptregs
546     x32     preadv2                 compat_sys_preadv64v2
547     x32     pwritev2                compat_sys_pwritev64v2
```

arch/x86/entry/syscalls/syscall_64.tbl

System Call

https://code.woboq.org/userspace/glibc/sysdeps/unix/sysv/linux/



User Mode

Application program

```
...
execve(path,
    argv, envp);
...
```

***glibc* wrapper function**

(sysdeps/unix/sysv/linux/execve.c)

```
execve(path, argv, envp)
{
  ...
  int 0x80
   (arguments: __NR_execve,
          path, argv, envp)
  ...
  return;
}
```

switch to kernel mode

switch to user mode

Kernel Mode

System call service routine

(arch/x86/kernel/process_32.c)

```
sys_execve()
{

  ...

  return error;
}
```

Trap handler

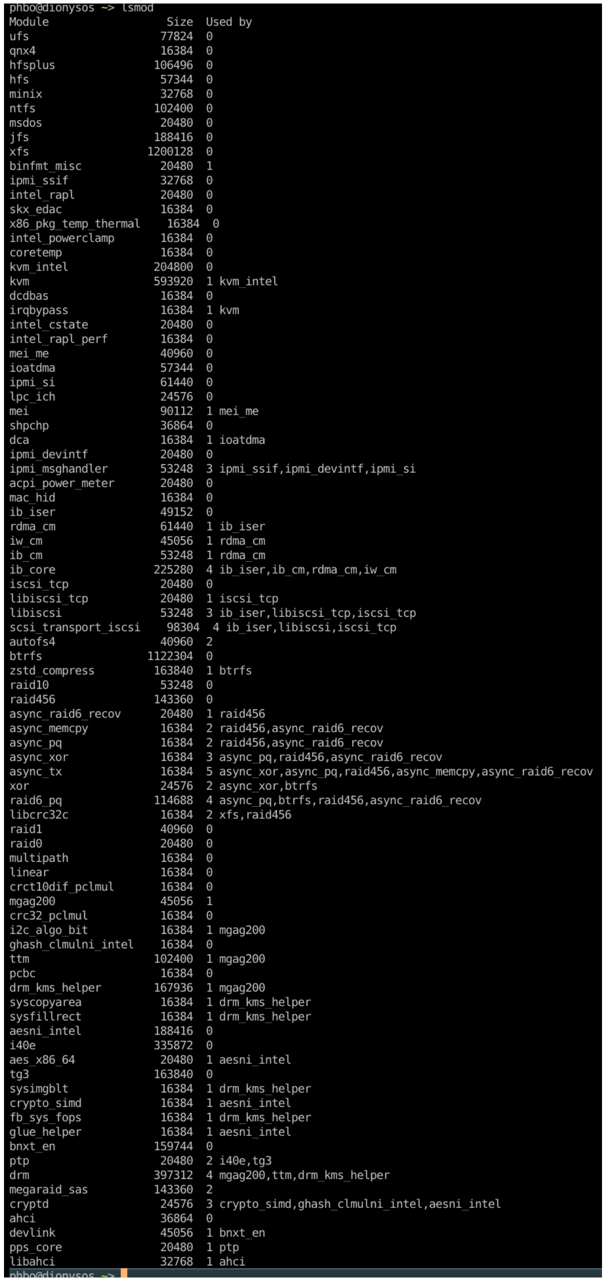
~~(arch/x86/kernel/entry_32.S)~~

```
system_call:

  ...

  call sys_call_table
        [__NR_execve]
  ...
```

arch/x86/kernel/entry/entry_32.S

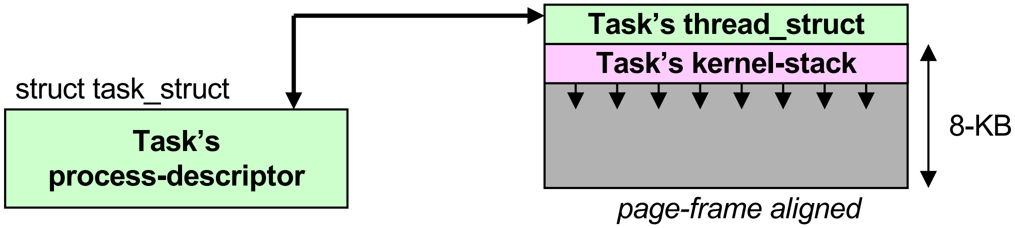
Loading Kernel Object

- Kernel modules are .ko files located in /lib/modules
- Commands:
  - lsmod: lists installed modules
  - Modprobe: installs module



```
phbo@dionysos ~> lsmod
Module                  Size  Used by
ufs                    77824  0
qnx4                   16384  0
hfsplus               106496  0
hfs                    57344  0
minix                  32768  0
ntfs                  102400  0
msdos                  20480  0
jfs                   188416  0
xfs                  1200128  0
binfmt_misc            20480  1
ipmi_ssif              32768  0
intel_rapl             20480  0
skx_edac               16384  0
x86_pkg_temp_thermal    16384  0
intel_powerclamp       16384  0
coretemp               16384  0
kvm_intel             204800  0
kvm                   593920  1 kvm_intel
dcdbas                 16384  0
irqbypass              16384  1 kvm
intel_cstate           20480  0
intel_rapl_perf        16384  0
mei_me                 40960  0
ioatdma                57344  0
ipmi_si                61440  0
lpc_ich                24576  0
mei                    90112  1 mei_me
shpchp                 36864  0
dca                    16384  1 ioatdma
ipmi_devintf           20480  0
ipmi_msghandler        53248  3 ipmi_ssif,ipmi_devintf,ipmi_si
acpi_power_meter       20480  0
mac_hid                16384  0
ib_iser                49152  0
rdma_cm                61440  1 ib_iser
iw_cm                  45056  1 rdma_cm
ib_cm                  53248  1 rdma_cm
ib_core               225280  4 ib_iser,ib_cm,rdma_cm,iw_cm
iscsi_tcp              20480  0
libiscsi_tcp           20480  1 iscsi_tcp
libiscsi               53248  3 ib_iser,libiscsi_tcp,iscsi_tcp
scsi_transport_iscsi    98304  4 ib_iser,libiscsi,iscsi_tcp
autofs4                40960  2
btrfs                1122304  0
zstd_compress         163840  1 btrfs
raid10                 53248  0
raid456               143360  0
async_raid6_recov      20480  1 raid456
async_memcpy           16384  2 raid456,async_raid6_recov
async_pq               16384  2 raid456,async_raid6_recov
async_xor              16384  3 async_pq,raid456,async_raid6_recov
async_tx               16384  5 async_xor,async_pq,raid456,async_memcpy,async_raid6_recov
xor                    24576  2 async_xor,btrfs
raid6_pq              114688  4 async_pq,btrfs,raid456,async_raid6_recov
libcrc32c              16384  2 xfs,raid456
raid1                  40960  0
raid0                  20480  0
multipath              16384  0
linear                 16384  0
crct10dif_pclmul       16384  0
mgag200                45056  1
crc32_pclmul           16384  0
i2c_algo_bit           16384  1 mgag200
ghash_clmulni_intel    16384  0
ttm                   102400  1 mgag200
pcbc                   16384  0
drm_kms_helper        167936  1 mgag200
syscopyarea            16384  1 drm_kms_helper
sysfillrect            16384  1 drm_kms_helper
aesni_intel           188416  0
i40e                  335872  0
aes_x86_64             20480  1 aesni_intel
tg3                   163840  0
sysimgblt              16384  1 drm_kms_helper
crypto_simd            16384  1 aesni_intel
fb_sys_fops            16384  1 drm_kms_helper
glue_helper            16384  1 aesni_intel
bnxt_en               159744  0
ptp                    20480  2 i40e,tg3
drm                   397312  4 mgag200,ttm,drm_kms_helper
megaraid_sas          143360  2
cryptd                 24576  3 crypto_simd,ghash_clmulni_intel,aesni_intel
ahci                   36864  0
devlink                45056  1 bnxt_en
pps_core               20480  1 ptp
libahci                32768  1 ahci
phbo@dionysos ~>
```

Processes and Threads



Process context:

- 8KB / process in kernel space to store process descriptor `task_struct` (/include/linux/sched.h).

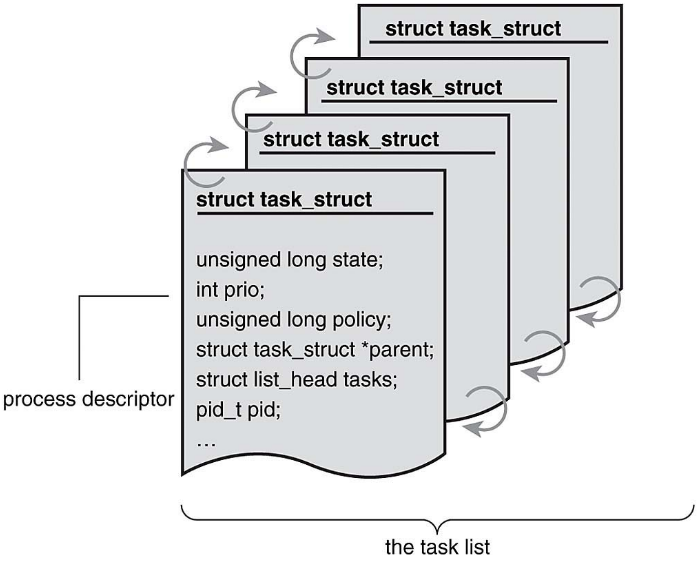
  State:

  ```
  #define TASK_RUNNING 0
  #define TASK_INTERRUPTIBLE 1
  #define TASK_UNINTERRUPTIBLE 2
  #define TASK_ZOMBIE 4
  #define TASK_STOPPED 8
  ```

  Process ID

  + virtual memory info, file system info, open files, signal handlers, …

- The thread of execution `thread_struct` (linux/arch/x86/include/asm/processor.h) PC, registers, Fault info,



Everything is a File Descriptor

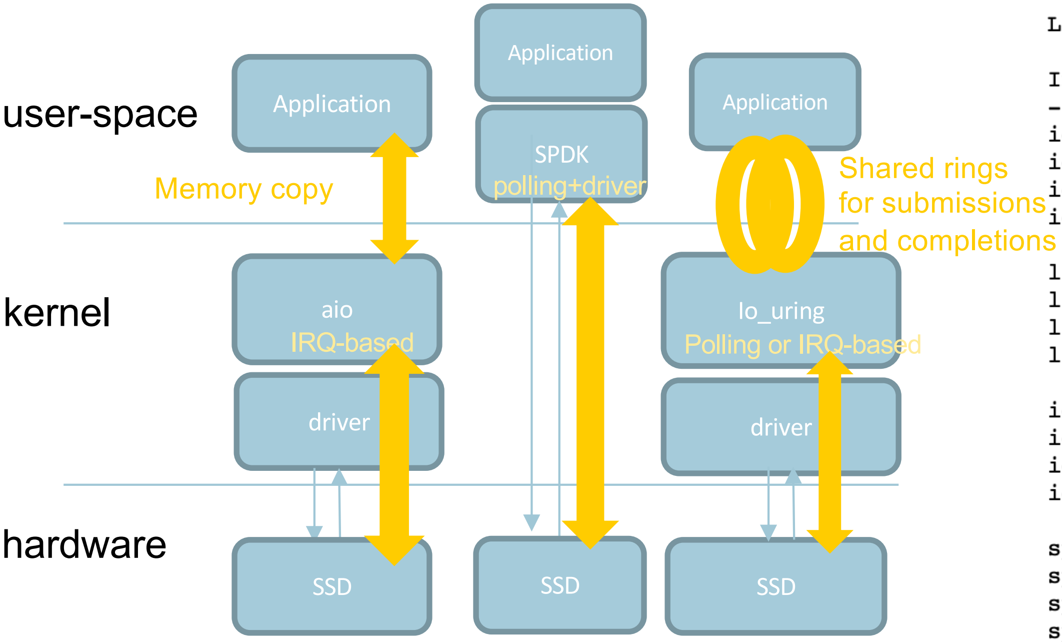
- Defining features of Unix, and its derivatives
  - File descriptor is a handle on a stream of bytes
  - Create/Delete, Open/close, Read/Write (seq/random)
- Linux uses the file system abstraction to provide access to hardware, configuration and debug information by exposing files that can be read and written.
  - Note that these special files are only exposed in file system name space - the files can be accessed like normal files but are not actually stored on any media.

Built-in file systems

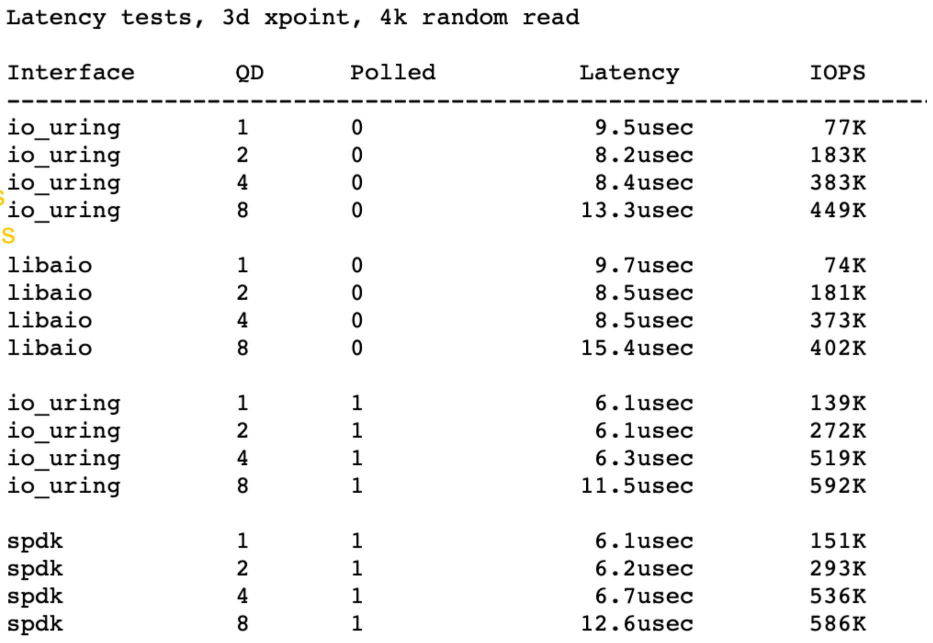
sys : a means to export kernel data structures, their attributes, and the linkages between them to userspace
dev : contains the special device files for all the devices
proc : more information to userspace (cmdline, version, devicetree)
debugfs : kernel to userspace debug information

e.g., /proc/version, /proc/info

EVERYTHING IS A FILE DESCRIPTOR



Source: Faster IO through io_uring & Efficient I/O with io_uring, J.Axboe



```
Latency tests, 3d xpoint, 4k random read

Interface       QD      Polled          Latency         IOPS
---------------------------------------------------------------
io_uring        1       0               9.5usec          77K
io_uring        2       0               8.2usec         183K
io_uring        4       0               8.4usec         383K
io_uring        8       0              13.3usec         449K

libaio          1       0               9.7usec          74K
libaio          2       0               8.5usec         181K
libaio          4       0               8.5usec         373K
libaio          8       0              15.4usec         402K

io_uring        1       1               6.1usec         139K
io_uring        2       1               6.1usec         272K
io_uring        4       1               6.3usec         519K
io_uring        8       1              11.5usec         592K

spdk            1       1               6.1usec         151K
spdk            2       1               6.2usec         293K
spdk            4       1               6.7usec         536K
spdk            8       1              12.6usec         586K
```

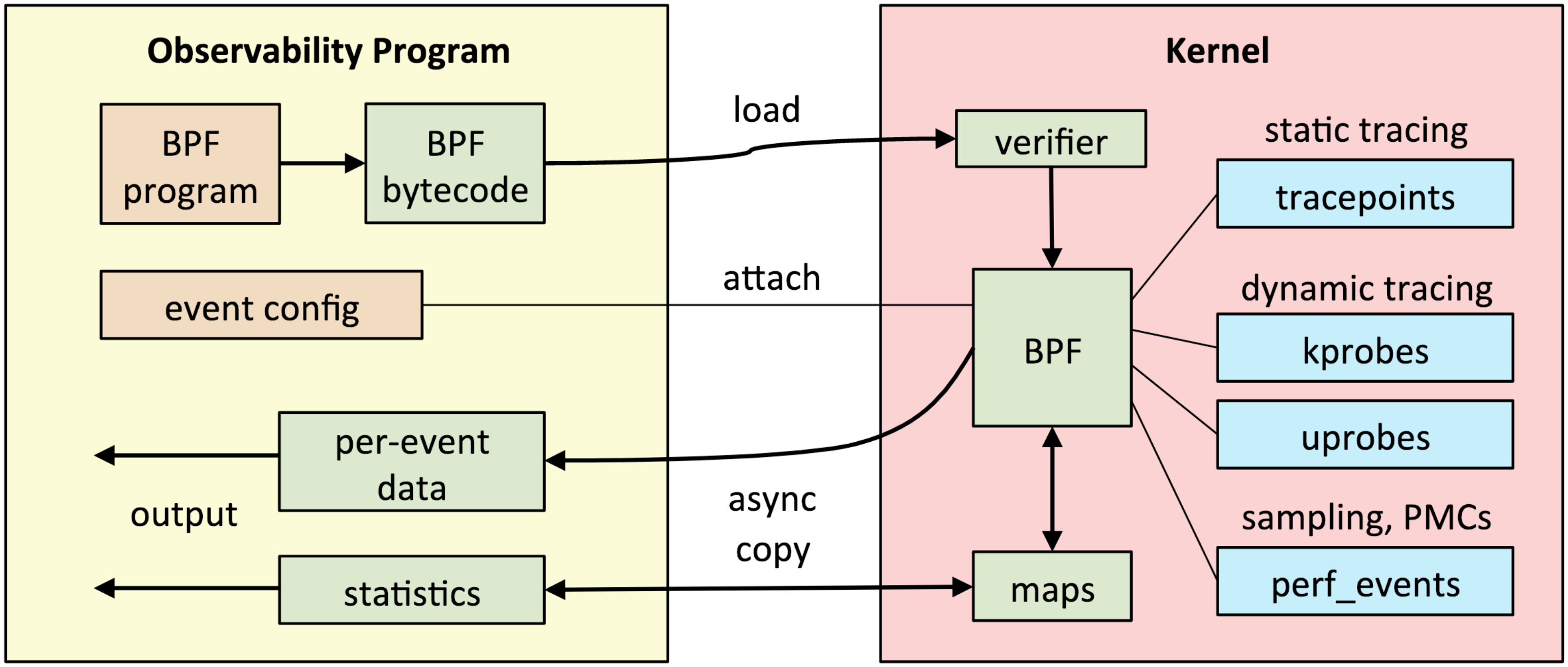
Source: Jens Axboe

Linux has learnt from research on Data plane OS!
OUT: POSIX. IN: zero copy and minimized synchronization overhead.

In-Kernel Virtualization

https://www.iovisor.org/technology/ebpf
http://www.brendangregg.com/blog/2018-10-08/dtrace-for-linux-2018.html

BPF for Tracing, Internals

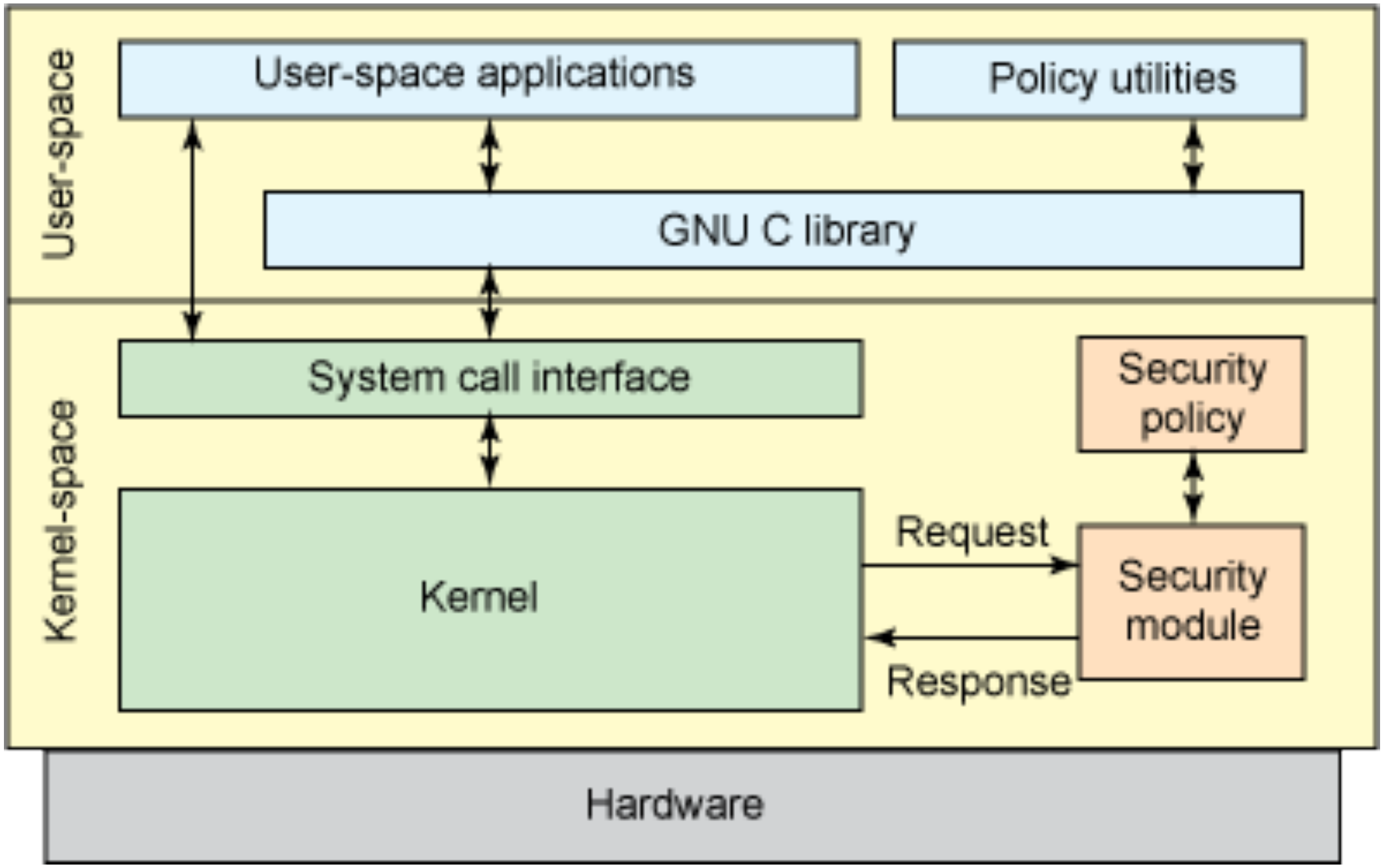


Enhanced BPF is also now used for SDNs, DDOS mitigation, intrusion detection, container security, …

Outline

- Context
- Kernel
- Community
- Loadable Modules
- Boot Process
- Key Concepts
- Linux Security Frameworks

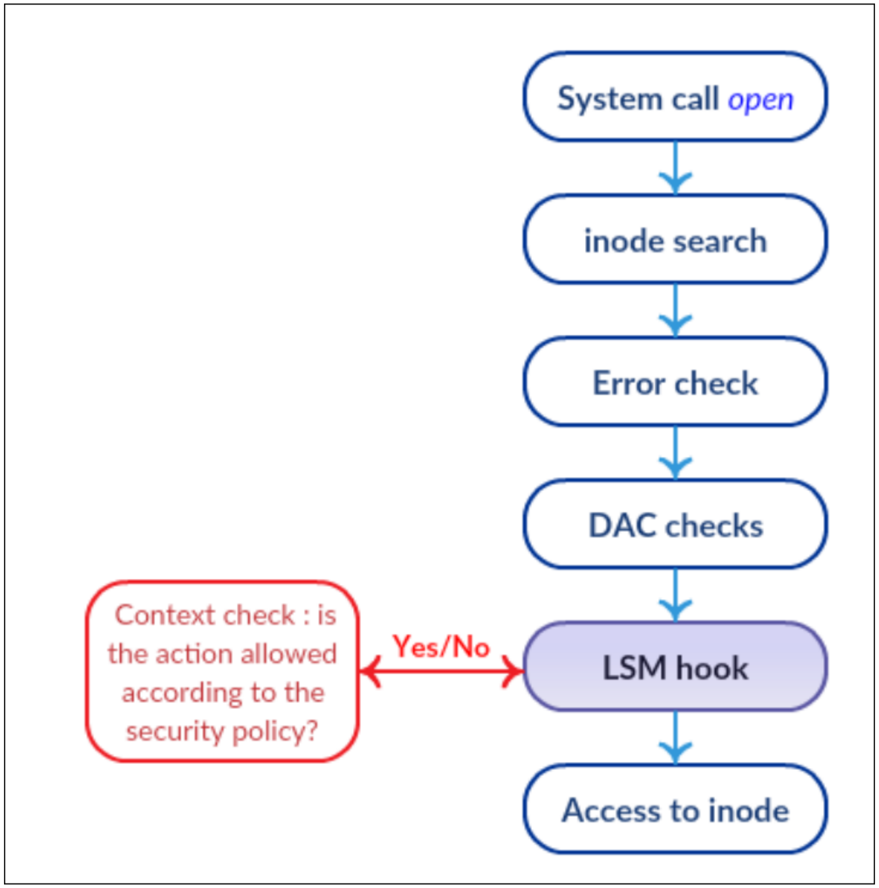
Linux Security Modules



LSM Hooks

- Each LSM hook is a function pointer in a global table, security_ops.
- Discretionary Access Control: restricts access to resources based on users and/or groups they belong
- Mandatory Access Control: programs can only do what they need to perform their tasks

  => Checks based on context



LSM Frameworks

General framework to control operations on kernel objects and a set of opaque security fields in kernel data structures for maintaining security attributes.

Used by **loadable kernel modules** to implement a given model of security.

- AppArmor
- SELinux
- Smack

Differences:

- Naming of kernel objects
- Definition of security fields
- Definition of security policies

https://ubuntu.com/tutorials/beginning-apparmor-profile-development#1-overview

Seccomp

- Leverages BPF
- MAC is a filter to prevent the calling process, or any descendants, to make a system call.
- Security policy defined in user space

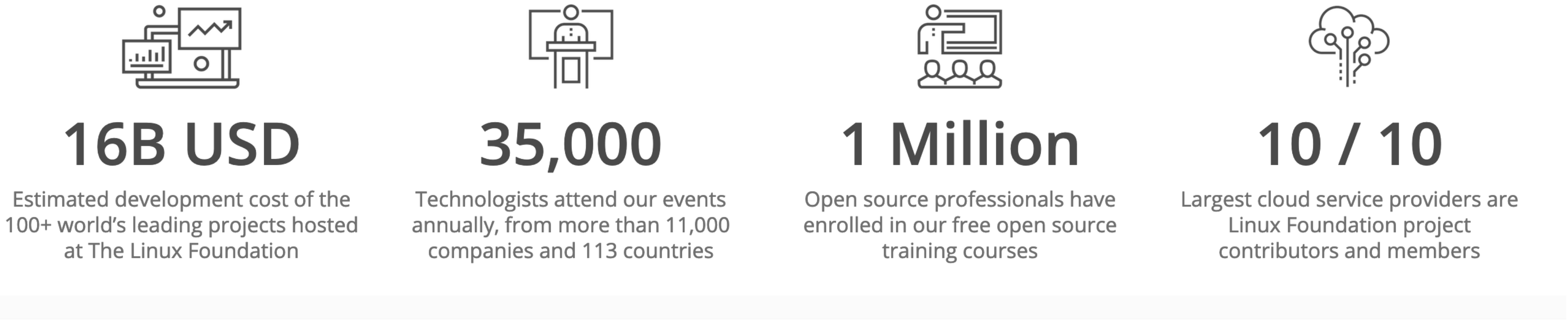
Seccomp filter cannot prevent a user process from opening files in only certain locations in the filesystem, like /etc/password.

Since seccomp filters cannot dereference pointers, they cannot compare the paths users pass as arguments to the open system call (like AppArmor) nor are they able to examine inodes to read security attributes attached to files (like SELinux).

Useful Resources

Follow Evolution on-line:

- https://www.kernel.org/
- https://github.com/torvalds/linux/
- https://elixir.bootlin.com/linux/latest/source
- https://lwn.net/
- https://lkml.org/



16B USD
Estimated development cost of the 100+ world's leading projects hosted at The Linux Foundation

35,000
Technologists attend our events annually, from more than 11,000 companies and 113 countries

1 Million
Open source professionals have enrolled in our free open source training courses

10 / 10
Largest cloud service providers are Linux Foundation project contributors and members