



THE LINUX OPERATING SYSTEM

A VERY BRIEF INTRODUCTION TO IT'S ARCHITECTURE

Important Notice : Courseware - Legal

This courseware is both the product of the author and of freely available opensource and/or public domain materials. Wherever external material has been shown, it's source and ownership have been clearly attributed. We acknowledge all copyrights and trademarks of the respective owners.

The contents of the **courseware PDFs are considered proprietary** and thus cannot be copied or reproduced in any form whatsoever without the explicit written consent of the author.

Only the programs - **source code** and binaries (where applicable) - that form part of this courseware, and that are made available to the participant, are released under the terms of the [permissive MIT license](#) [1].

Under the terms of the MIT License, you can certainly use the source code provided here; you must just attribute the original source (author of this courseware and/or other copyright/trademark holders).

VERY IMPORTANT :: Before using this source(s) in your project(s), you ***MUST*** check with your organization's legal staff that it is appropriate to do so.

The courseware PDFs are ***not*** under the MIT License, they are to be kept confidential, non-distributable without consent, for your private internal use only.

The duration, contents, content matter, programs, etc. contained in this courseware and companion participant VM are subject to change at any point in time without prior notice to individual participants.

Care has been taken in the preparation of this material, but there is no warranty, expressed or implied of any kind, and we can assume no responsibility for any errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

2000-2020 Kaiwan N Billimoria
kaiwanTECH, Bangalore, India.

kaiwanTECH Linux OS Corporate Training Programs
Please do check out our current offering of world-class, seriously-valuable, high on returns, technical Linux OS corporate training programs here: http://bit.ly/ktcorp

Linux / Unix Architecture

[Source](#)

“In the old days, you had a processor and it executed instructions. When an interrupt occurred, the processor would save its current state and then branch to a specific place in order to service the interrupt. Thus, essentially, the processor had two 'modes' - dealing with an interrupt, and not.

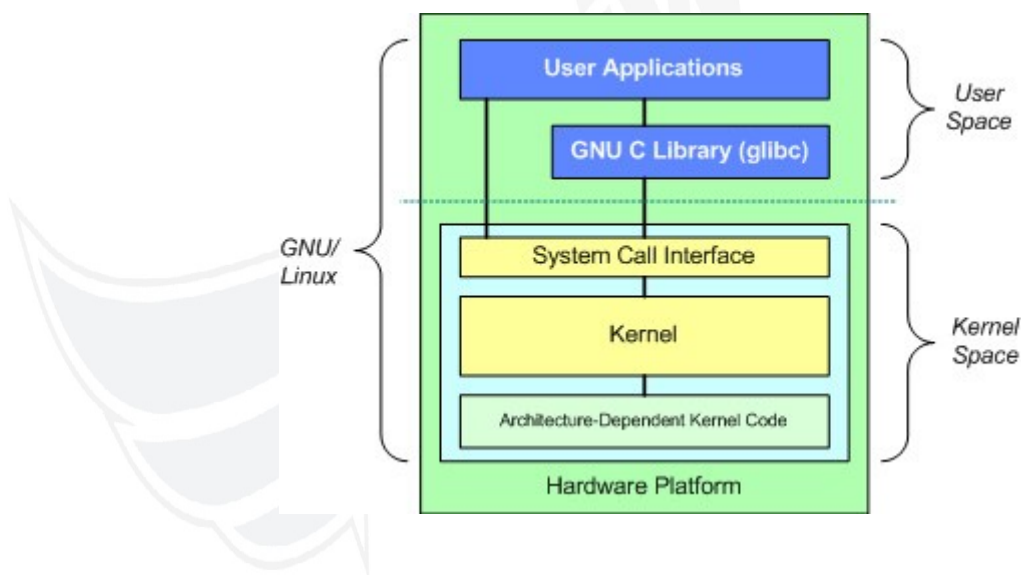
Fast forward a few decades, processors are much more capable and something of great importance is the ability to multitask, to run numerous programs at the same time. Technically this isn't possible, the processor has one data bus and one address bus, and however many cores are inside it, each core can only do one thing at a time. However by breaking a program's execution into tiny chunks, the processor can switch between them many times a second, providing the illusion that they are all running at the same time.

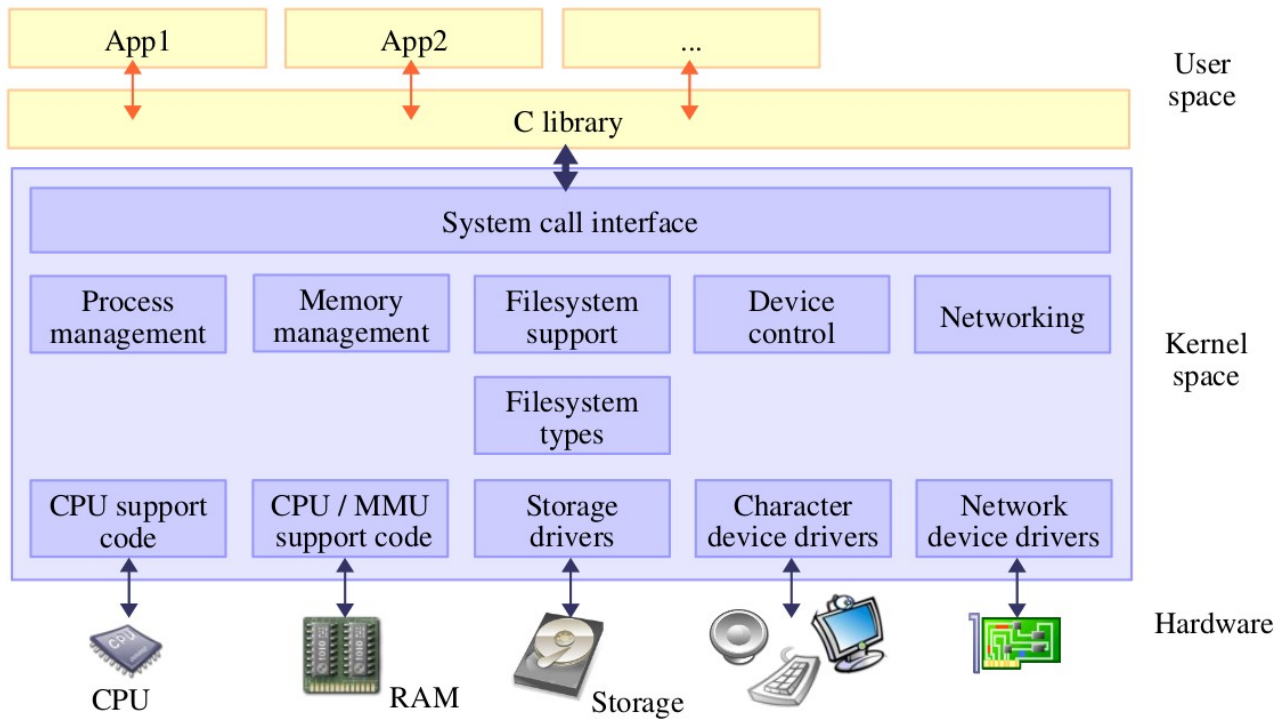
This brings with it a number of additional requirements, namely that

- one program should not be able to mess around with the memory used by another program
- furthermore, *none* of the mere programs should be able to mess around with the operating system or the machine's hardware

This is managed, in part, by the use of an MMU or other memory management system, and in part by the use of *privilege*. It is the *processor mode that provides the desired level of privilege. ...*”

Simplified System Architecture

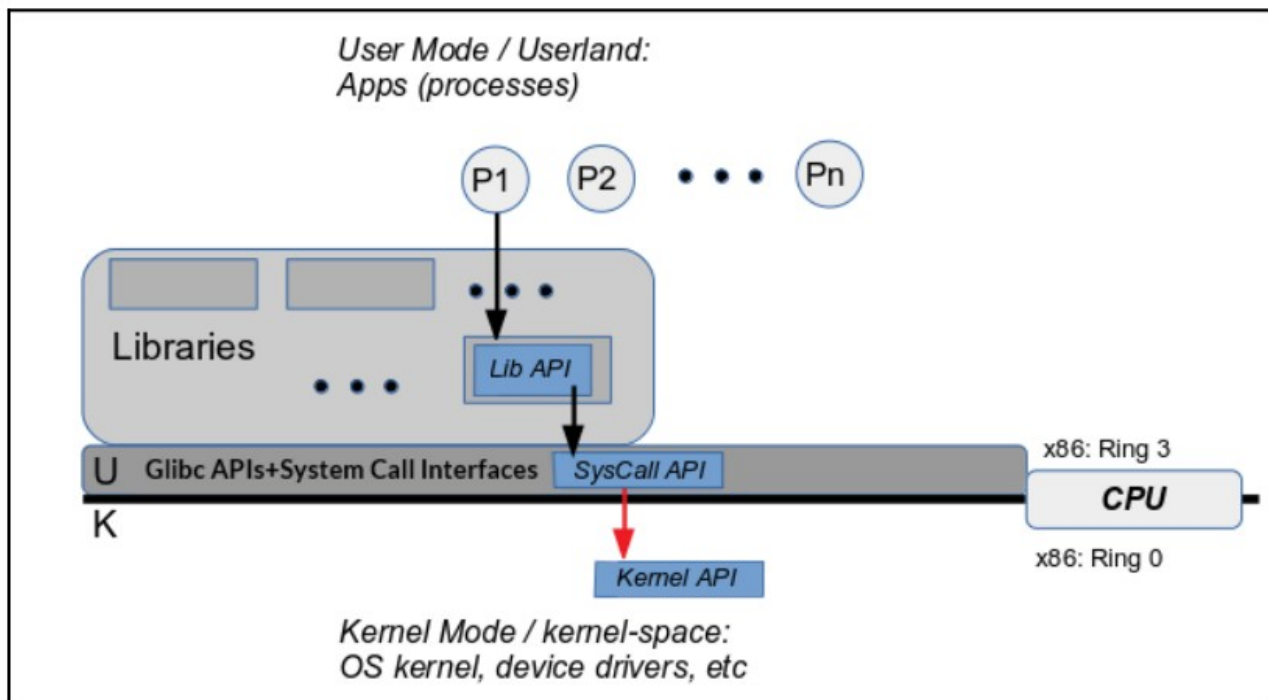


More Detailed System Architecture

<< Above Pic: © Copyright 2006 2004, Michael Opdenacker , © Copyright 2004 2008 Codefidence Ltd. >>

<<

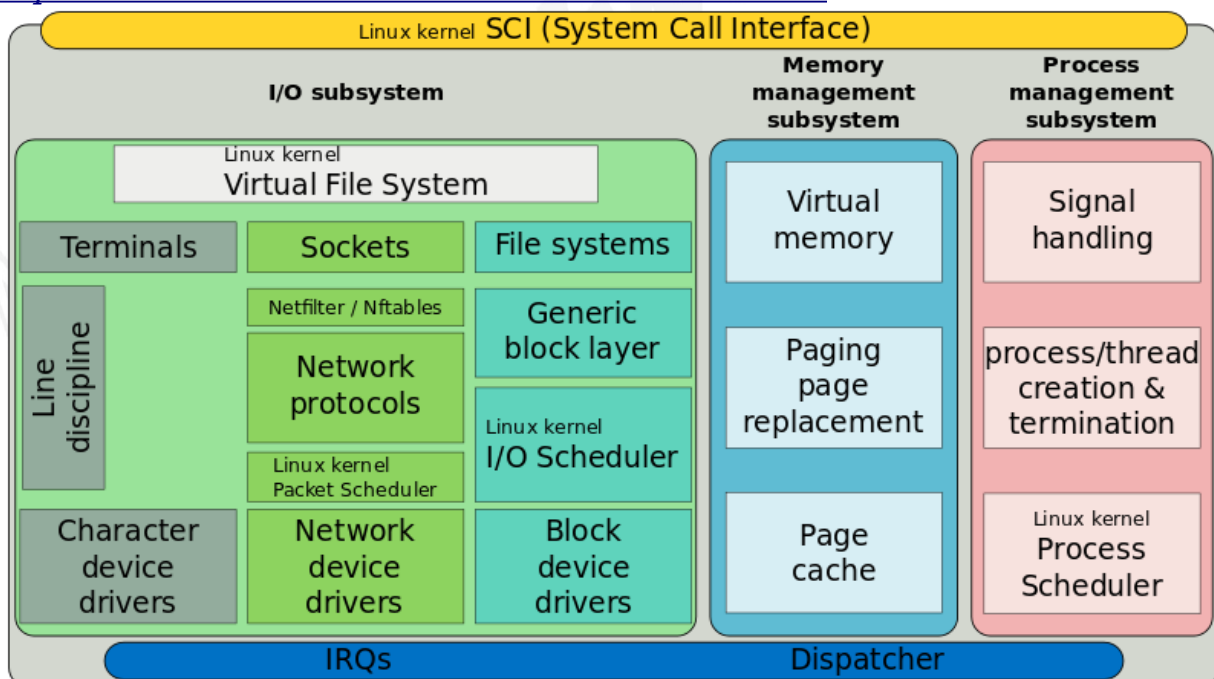
From: [*'Hands-On System Programming with Linux', Kaiwan N Billimoria, Packt:*](#)



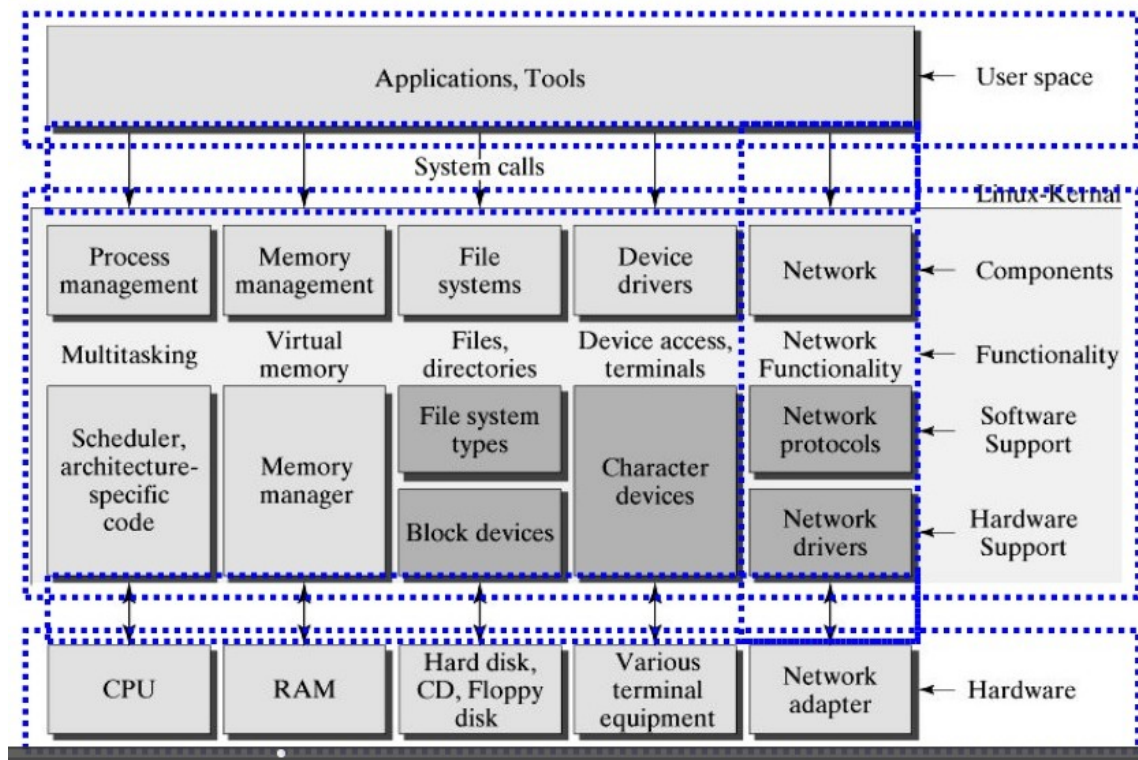
>>

<<

[*Simplified Structure of the Linux Kernel : Wikimedia*](#)



Yet another architecture diagram [[Source](#)]:



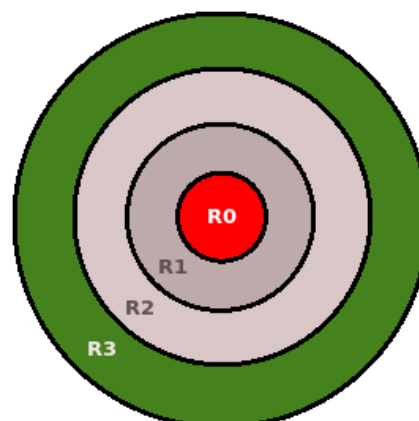
>>

Privilege Levels in Different CPU Architectures

x86

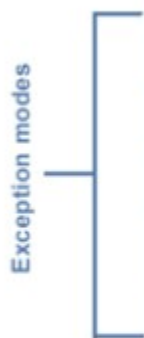
Discussion

- The SCI - System Call Interface - Layer
- CPU Privilege Levels
 - User Mode
 - Kernel (Supervisor) Mode
- *Intel/AMD: Ring 3 => User, Ring 0 => OS*



ARM (Aarch32) CPU Modes

7 modes, 6 of which are privileged



Mode	Description	
Supervisor (SVC)	Entered on reset and when a Software Interrupt instruction (SWI) is executed	Privileged modes
FIQ	Entered when a high priority (fast) interrupt is raised	
IRQ	Entered when a low priority (normal) interrupt is raised	
Abort	Used to handle memory access violations	
Undef	Used to handle undefined instructions	
System	Privileged mode using the same registers as User mode	Unprivileged mode
User	Mode under which most Applications / OS tasks run	

Aarch64 (ARM-64 / ARMv8)

- Modes replaced by processor **Exception Levels (Els)**
- Four ELs (*lowest to highest privilege*)
 - **EL0** : Normal user applications
 - **EL1** : Operating System (OS) kernel typically described as *privileged*
 - **EL2** : Hypervisor [optional]
 - **EL3** : Low-level firmware, including the Secure Monitor
- Here ELs determine the CPU privilege level (just as Modes do in Aarch32)
- Exception: in-kernel hypervisors (such as KVM) – operate across both EL1 and EL2.

System Calls

Architecture	Machine Instruction(s)	Syscall # Register
Intel x86[_64]	int \$0x80 syscall	EAX / RAX
ARM	SWI	r0
MIPS	syscall	\$v0

Details**[FYI/Optional]**

From *man syscall(2)*:

--snip--

Architecture calling conventions

Every architecture has its own way of invoking and passing arguments to the kernel. The details for various architectures are listed in the two tables below.

The first table lists the instruction used to transition to kernel mode (which might not be the fastest or best way to transition to the kernel, so you might have to refer to *vdso(7)*), the register used to indicate the system call number, the register used to return the system call result, and the register used to signal an error.

arch/ABI	instruction	syscall #	retval	error	Notes
alpha	callsys	v0	a0	a3	[1]
arc	trap0	r8	r0	-	
arm/OABI	swi NR	-	a1	-	[2]
arm/EABI	swi 0x0	r7	r0	-	
arm64	svc #0	x8	x0	-	
blackfin	excpt 0x0	P0	R0	-	
i386	int \$0x80	eax	eax	-	
ia64	break 0x100000	r15	r8	r10	[1]
m68k	trap #0	d0	d0	-	
microblaze	brki r14,8	r12	r3	-	
mips	syscall	v0	v0	a3	[1]
nios2	trap	r2	r2	r7	
parisc	ble 0x100(%sr2, %r0)	r20	r28	-	
powerpc	sc	r0	r3	r0	[1]
s390	svc 0	r1	r2	-	[3]
s390x	svc 0	r1	r2	-	[3]
superh	trap #0x17	r3	r0	-	[4]
sparc/32	t 0x10	g1	o0	psr/csr	[1]
sparc/64	t 0x6d	g1	o0	psr/csr	[1]
tile	swint1	R10	R00	R01	[1]
x86_64	syscall	rax	rax	-	[5]
x32	syscall	rax	rax	-	[5]
xtensa	syscall	a2	a2	-	

--snip--

Implementation of System Calls within Android's (AOSP) Bionic (it's libc replacement)

Lets take the common *getpid(2)* systme call as an example:

x86_64

<AOSP>/bionic/libc/arch-x86_64/syscalls/__getpid.S


```
/* Generated by gensyscalls.py. Do not edit. */
```

```
#include <private/bionic_asm.h>
```

```
ENTRY(__getpid)
    movl    $__NR_getpid, %eax
    syscall
    cmpq    $-MAX_ERRNO, %rax
    jb      1f
    negl    %eax
    movl    %eax, %edi
    call    __set_errno_internal
1:
    ret
END(__getpid)
.hidden __getpid
```

ARM (Aarch32)

[<AOSP>/bionic/libc/arch-arm/syscalls/__getpid.S](#)

```
/* Generated by gensyscalls.py. Do not edit. */
```

```
#include <private/bionic_asm.h>
```

```
ENTRY(__getpid)
    mov     ip, r7
    .cfi_register r7, ip
    ldr     r7, =__NR_getpid
    swi     #0
    mov     r7, ip
    .cfi_restore r7
    cmn     r0, #(MAX_ERRNO + 1)
    bxls    lr
    neg     r0, r0
    b       __set_errno_internal
END(__getpid)
```

ARM-64 (Aarch64)

[<AOSP-Pie 9.0.0-r3>/bionic/libc/arch-arm64/syscalls/_getpid.S](#)

```
/* Generated by gensyscalls.py. Do not edit. */
```

```
#include <private/bionic_asm.h>
```

```
ENTRY(__getpid)
    mov     x8, __NR_getpid
    svc     #0

    cmn     x0, #(MAX_ERRNO + 1)
    cneg     x0, x0, hi
    b.hi     __set_errno_internal

    ret
END(__getpid)
.hidden __getpid
```

MIPS-32

<AOSP>/bionic/libc/arch-mips/syscalls/__getpid.S

/* Generated by gensyscalls.py. Do not edit. */

#include <private/bionic_asm.h>

```
ENTRY(__getpid)
    .set noreorder
    .cpld $t9
    li $v0, __NR_getpid
    syscall
    bnez $a3, 1f
    move $a0, $v0
    j $ra
    nop
1:
    la $t9, __set_errno_internal
    j $t9
    nop
    .set reorder
END(__getpid)
```

Microsoft releases its first Linux product

For the first time, Microsoft has released its own Linux kernel in a new Linux-based product: Azure Sphere. [Apr 17 2018]

“Well, it's finally happened. [Microsoft has released a product containing its own Linux kernel: Azure Sphere](#). It's not MS-Linux or Linux Windows, but it's still remarkable.

[Azure Sphere](#) is a software and hardware stack designed to secure edge devices. It includes microcontrollers, "Azure Sphere Security Service" and, the really interesting component, the Linux-based Azure Sphere operating system.

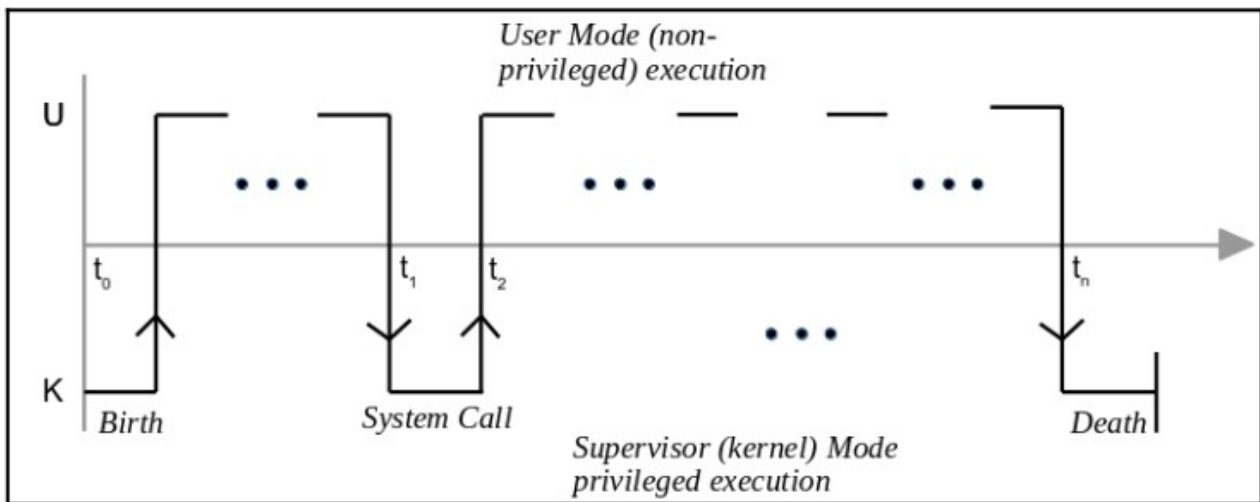
Microsoft made no bones about it. Microsoft President Brad Smith introduced Azure Sphere saying, "After 43 years, this is the first day that we are announcing, and will distribute, a custom Linux kernel."

In technology's terms that's "My, how things have changed". Even former Microsoft CEO Steve Ballmer, [who once called Linux a cancer](#), can see that Microsoft now needs Linux.

Linus Torvalds once said, "If Microsoft ever does applications for Linux it means I've won." He's won.

[...]"

Flow of a Process – Birth to Death – between privilege levels



From: [*'Hands-On System Programming with Linux', Kaiwan N Billimoria, Packt:*](#)

- The following system instructions are used to control those functions of the processor that are provided to support for operating systems and executives.
 - Manipulate memory management register
 - **LGDT, LLDT, LTR, LIDT**, SGDT, SLDT, SIDT, STR
 - Load and store control registers
 - **MOV {CR0~CR4}, CLI, STI**
 - Invalidate Cache and TLB
 - **INVD, WBINVD, INVLPG**
 - Performance monitoring
 - **RDPMC, RDTSC, RDTSCP**
 - Fast System Call
 - SYSENTER, SYSEXIT
 - Pointer Validation
 - LAR, LSL, VERR, VERW, ARPL
 - Misc.
 - LOCK, **CLTS, HLT**

Privileged instructions in **red** which can be executed only in ring 0.

[Source](#)

Architecture



Various layers within Linux, also showing separation between the [userland](#) and [kernel space](#)

User mode	User applications	For example, bash , LibreOffice , Apache OpenOffice , Blender , 0 A.D. , Mozilla Firefox , etc.				
	Low-level system components:	System daemons: systemd , runit , logind , networkd , soundd , ...	Windowing system: X11 , Wayland , Mir , SurfaceFlinger (Android)	Other libraries: GTK+ , Qt , EFL , SDL , SFML , FLTK , GNUstep , etc.		Graphics: Mesa , AMD Catalyst , ...
	C standard library	open() , exec() , sbrk() , socket() , fopen() , calloc() , ... (up to 2000 subroutines) glibc aims to be POSIX/SUS -compatible, uClibc targets embedded systems, bionic written for Android , etc.				
Kernel mode	Linux kernel	stat , splice , dup , read , open , ioctl , write , mmap , close , exit , etc. (about 380 system calls) The Linux kernel System Call Interface (SCI, aims to be POSIX/SUS -compatible)				
		Process scheduling subsystem	IPC subsystem	Memory management subsystem	Virtual files subsystem	Network subsystem
		Other components: ALSA , DRI , evdev , LVM , device mapper , Linux Network Scheduler , Netfilter Linux Security Modules : SELinux , TOMOYO , AppArmor , Smack				
Hardware (CPU, main memory, data storage devices, etc.)						


...



Monolithic Kernel

Googling “monolithic meaning” gets one this:

monolith

/ˈmɒn(ə)lɪθ/ 

noun

noun: **monolith**; plural noun: **monoliths**

1. a large single upright block of stone, especially one shaped into or serving as a pillar or monument.
"we passed Stonehenge, the strange stone monoliths silhouetted against the horizon"
synonyms: standing stone, menhir, sarsen (stone), megalith
 - a very large and characterless building.
"the 72-storey monolith overlooking the waterfront"
 - a large block of concrete sunk in water, e.g. in the building of a dock.
2. a large, impersonal political, corporate, or social structure regarded as indivisible and slow to change.
"independent voices have been crowded out by the media monoliths"

Origin

GREEK

monos
single

GREEK

monolithos

FRENCH

monolithe

monolith

mid 19th century

GREEK

lithos
stone

mid 19th century: from French *monolithe*, from Greek *monolithos*, from *monos* 'single' + *lithos* 'stone'.



Stonehenge

Linux being “monolithic” implies that:

- the entire kernel code runs in a separate address space called “kernel-space”
- this kernel-space is shared by all usermode processes

[Source](#) [below]

[...]

Linux is a [monolithic kernel](#). [Device drivers](#) and kernel extensions run in [kernel space](#) (ring 0 in many [CPU architectures](#)), with full access to the hardware, although some exceptions run in [user space](#), for example filesystems based on [FUSE](#). The [graphics system](#) most people use with Linux doesn't run in the kernel, in contrast to that found in [Microsoft Windows](#). Unlike standard monolithic kernels, device drivers are easily configured as [modules](#), and loaded or unloaded while running the system. Also unlike standard monolithic kernels, device drivers can be pre-empted under certain conditions. This latter feature was added to handle [hardware interrupts](#) correctly, and to improve support for [symmetric multiprocessing](#).[\[citation needed\]](#) By choice, the Linux kernel has no [Binary Kernel Interface](#).[\[45\]](#)

The hardware is also incorporated into the file hierarchy. Device drivers interface to user applications via an entry in the [/dev](#)[\[46\]](#) and/or [/sys](#) directories. Process information as well is mapped to the file system through the [/proc](#) directory.[\[46\]](#)

Linux supports true [preemptive multitasking](#) (both in [user mode](#) and [kernel mode](#)), [virtual memory](#), [shared libraries](#), [demand loading](#), shared [copy-on-write](#) executables (via [KSM](#)), [memory management](#), the [Internet protocol suite](#), and [threading](#).

...

[Source](#)

Monolithic Kernel

A monolithic kernel is an operating system architecture where the entire operating system is working in [kernel space](#) and is alone in [supervisor mode](#). The monolithic model differs from other operating system architectures (such as the [microkernel](#) architecture)[\[1\]\[2\]](#) in that it alone defines a high-level virtual interface over computer hardware. A set of primitives or [system calls](#) implement all operating system services such as [process](#) management, [concurrency](#), and [memory management](#). Device drivers can be added to the kernel as [modules](#).

Monolithic architecture examples

Unix kernels

[_BSD](#)
[_FreeBSD](#)
[_NetBSD](#)
[_OpenBSD](#)
[_Solaris 1](#) / [SunOS 1.x-4.x](#)

UNIX System V

[_AIX](#)
[_HP-UX](#)

Unix-like kernels

[_Linux](#)

DOS

[_DR-DOS](#)
[_MS-DOS](#)

____ Microsoft [Windows 9x](#) series ([95](#), [98](#), [Windows 98SE](#), [Me](#))

OpenVMS

[XTS-400](#)



An obelisk – a monolith

Microkernel

In [computer science](#), a **microkernel** (also known as μ -kernel) is the near-minimum amount of software that can provide the mechanisms needed to implement an [operating system](#) (OS). These mechanisms include low-level [address space](#) management, [thread](#) management, and [inter-process communication](#) (IPC). If the hardware provides multiple [rings](#) or [CPU modes](#), the microkernel is the only software executing at the most privileged level (generally referred to as [supervisor or kernel mode](#)).

[[citation needed](#)] Traditional operating system functions, such as [device drivers](#), [protocol stacks](#) and [file systems](#), are removed from the microkernel to run in [user space](#). [[citation needed](#)] In source code size, microkernels tend to be under 10,000 lines of code, as a general rule. [MINIX](#)'s kernel, for example has fewer than 6,000 lines of code. [[1](#)]
 Example : Minix, QNX, VxWorks. << OLD >>

See the [Wikipedia page on Category:Microkernels](#); there are over 70 microkernel OS's here.

<<

[**OPTIONAL / FYI**]

[Source: 5 Major Software Architecture Patterns](#)

“Microkernel Pattern

The microkernel architectural pattern is also referred to as a plug-in architectural pattern. It is typically used when software teams create systems with interchangeable components.

It applies to software systems that must be able to adapt to changing system requirements. It separates a minimal functional core from extended functionality and customer-specific parts. It'll also serve as a socket for plugging in these extensions and coordinating their collaboration.

--snip--

The microkernel architecture pattern consists of two types of architecture components: a core system and plug-in modules. Application logic is divided between independent plug-in modules and the basic core system, providing extensibility, flexibility, and isolation of application features and custom processing logic. And the core system of the microkernel architecture pattern traditionally contains only the minimal functionality required to make the system operational.

Perhaps the best example of the microkernel architecture is the *Eclipse IDE*. Downloading the basic Eclipse product provides you little more than an editor. However, once you start adding plug-ins, it becomes a highly customizable and useful product. ...”

>>

Hybrid

A **hybrid kernel** is a [kernel](#) architecture based on combining aspects of [microkernel](#) and [monolithic kernel](#) architectures used in [computer operating systems](#). The traditional kernel categories are [monolithic kernels](#) and [microkernels](#) (with [nanokernels](#) and [exokernels](#) seen as more extreme versions of microkernels). The category is **controversial** due to the similarity to monolithic kernel; the term has been dismissed by [Linus Torvalds](#) as simple marketing. [\[1\]](#)

The idea behind this category is to have a kernel structure similar to a microkernel, but implemented in terms of a monolithic kernel. In contrast to a microkernel, all (or nearly all) operating system services are in [kernel space](#). While there is no performance overhead for message passing and context switching between kernel and user mode, as in [monolithic kernels](#), there are no reliability benefits of having services in [user space](#), as in [microkernels](#).

Implementations

[BeOS](#) kernel

[Haiku](#) kernel

[Syllable](#)

[BSD](#)-based

[DragonFly BSD](#) (first non-[Mach](#) BSD OS to use a hybrid kernel)

[XNU](#) kernel (core of [Darwin](#), used in [Mac OS X](#) and [iOS](#))

[NetWare](#) kernel [\[7\]](#)

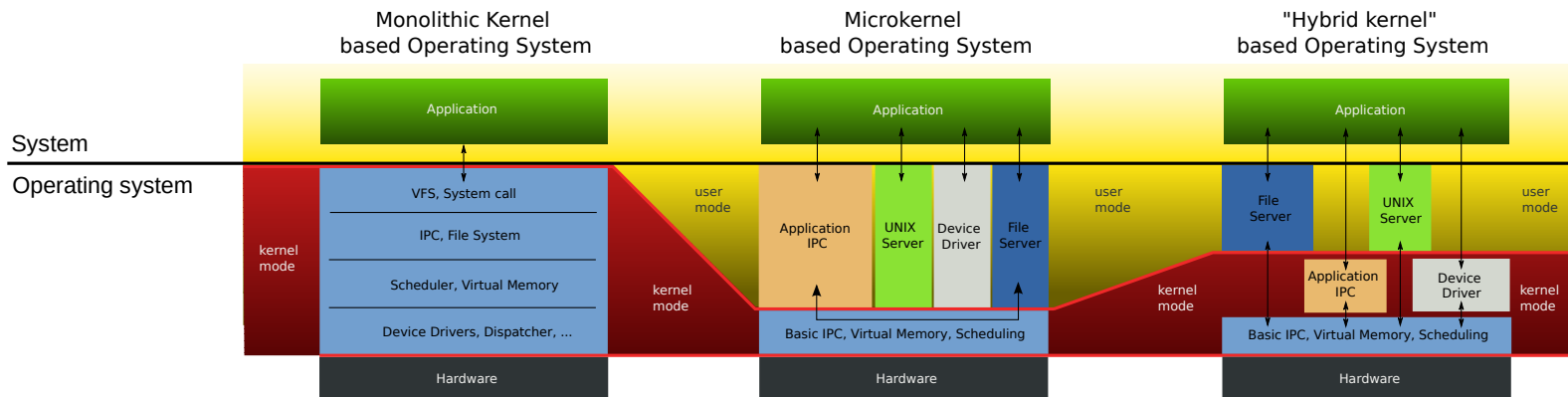
[Inferno](#) kernel

NT kernel (used in [Windows NT 3.1](#), [Windows NT 3.5](#), [Windows NT 4.0](#), [Windows 2000](#), [Windows Server 2003](#), [Windows XP](#), [Windows Vista](#), [Windows Server 2008](#), [Windows 7](#), [Windows Server 2008 R2](#), [Windows 8](#), and [Windows Server 2012](#))

... One prominent example of a hybrid kernel is the [Microsoft NT kernel](#) that powers all operating systems in the [Windows NT](#) family, **up to and including Windows 10** and [Windows Server 2012](#), and powers [Windows Phone 8](#), [Windows Phone 8.1](#), and [Xbox One](#). ...

<< [Find more overview information on the Windows NT design here](#), section "NT kernel" >>

[ReactOS](#) kernel



Ref:

[How does Linux kernel compare to microkernel architectures?](#)

[Why is Linux called a monolithic kernel?](#)

Instructor : mention what Process and Interrupt Contexts are.

[OPTIONAL / FYI]

An FAQ regarding keeping track of Linux kernel Changes

The Linux kernel is a very fast moving target: things change, quite rapidly at times, new enhancements and features get merged, kernel internal APIs / ABIs get deprecated, etc etc.

An example: a kernel module which built and worked perfectly on an earlier kernel fails to even compile on a recent (4.10) kernel:

```
[...]
$ make
make -Wall -C /lib/modules/4.10.0-33-generic/build M=<...> modules
make[1]: Entering directory '/usr/src/linux-headers-4.10.0-33-generic'
Building with KERNELRELEASE = 4.10.0-33-generic
  CC [M]  <...>/2nf.o
<...>/2nf.c: In function 'reg_nf':
<...>/2nf.c:162:10: error: 'struct nf_hook_ops' has no member named 'owner'
  psNFHook->owner = THIS_MODULE;
             ^~
<...>/2nf.c: In function 'nf_init':
<...>/2nf.c:230:44: error: passing argument 3 of 'reg_nf' from incompatible pointer type
[-Werror=incompatible-pointer-types]
  reg_nf(&nfhk_in_pre, NF_INET_PRE_ROUTING, nfhook_in_pre, PF_INET, NF_IP_PRI_FIRST);
                                             ^~~~~~
<...>/2nf.c:159:20: note: expected 'unsigned int (*)(void *, struct sk_buff *, const
struct nf_hook_state *)' but argument is of type 'unsigned int (*)(const struct
nf_hook_ops *, struct sk_buff *, const struct net_device *, const struct net_device *, int
*)(struct sk_buff *)'
  static inline void reg_nf(struct nf_hook_ops *psNFHook, int hooknum,
                         ^~~~~~
<...>/2nf.c:231:43: error: passing argument 3 of 'reg_nf' from incompatible pointer type
[-Werror=incompatible-pointer-types]
  reg_nf(&nfhk_local_in, NF_INET_LOCAL_IN, nfhook_local_in, PF_INET, NF_IP_PRI_FIRST);
                                             ^~~~~~
<...>/2nf.c:159:20: note: expected 'unsigned int (*)(void *, struct sk_buff *, const
struct nf_hook_state *)' but argument is of type 'unsigned int (*)(const struct
nf_hook_ops *, struct sk_buff *, const struct net_device *, const struct net_device *, int
*)(struct sk_buff *)'
  static inline void reg_nf(struct nf_hook_ops *psNFHook, int hooknum,
                         ^~~~~~
[...]

cc1: some warnings being treated as errors
scripts/Makefile.build:301: recipe for target '<...>/2nf.o' failed
make[2]: *** [<...>/2nf.o] Error 1
Makefile:1524: recipe for target '_module_<...>' failed
make[1]: *** [_module_<...>] Error 2
make[1]: Leaving directory '/usr/src/linux-headers-4.10.0-33-generic'
Makefile:16: recipe for target 'build' failed
make: *** [build] Error 2
$
```

2.6.36 : the ioctl() signature changes

[below: code from a device driver taking this into account]

```
#include <linux/version.h>
[...]
```

```
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,36)
static long rwmem_iocctl(struct file *filp, unsigned int cmd, unsigned long arg)
#else
static int rwmem_iocctl(struct inode *ino, struct file *filp, unsigned int cmd, unsigned long arg)
#endif
[...]
```

Why are the kernel APIs “unstable”?

From: <https://www.kernel.org/doc/html/latest/process/1.Intro.html#the-importance-of-getting-code-into-the-mainline>

... While kernel developers strive to maintain a stable interface to user space, the internal kernel API is in constant flux. The lack of a stable internal interface is a deliberate design decision; it allows fundamental improvements to be made at any time and results in higher-quality code. But one result of that policy is that any out-of-tree code requires constant upkeep if it is to work with new kernels. Maintaining out-of-tree code requires significant amounts of work just to keep that code working.

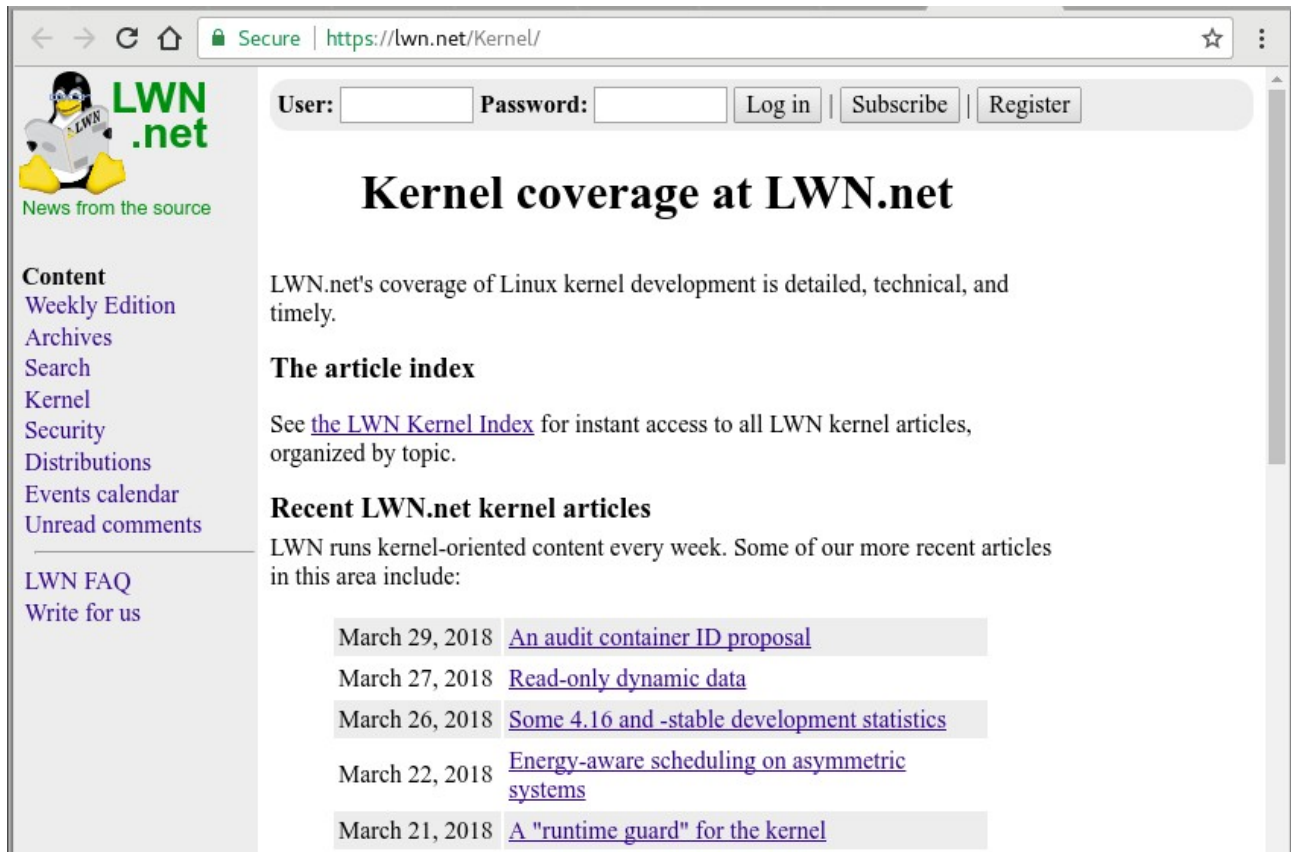
Code which is in the mainline, instead, does not require this work as the result of a simple rule requiring any developer who makes an API change to also fix any code that breaks as the result of that change. So code which has been merged into the mainline has significantly lower maintenance costs. ...

< many more points follow as well >

How can one sanely keep track of all changes?

The knee-jerk answer: follow the LKML (Linux Kernel Mailing List).
But “sanely”?
:-)

Read the
<https://lwn.net/Kernel/>



The screenshot shows the LWN.net website's 'Kernel' section. The browser address bar displays 'https://lwn.net/Kernel/'. The page features a navigation sidebar on the left with links like 'Weekly Edition', 'Archives', and 'Search'. The main content area is titled 'Kernel coverage at LWN.net' and includes a login/register section, a description of the site's coverage, a link to the 'LWN Kernel Index', and a list of recent kernel articles with dates and titles.

Kernel coverage at LWN.net

LWN.net's coverage of Linux kernel development is detailed, technical, and timely.

The article index

See [the LWN Kernel Index](#) for instant access to all LWN kernel articles, organized by topic.

Recent LWN.net kernel articles

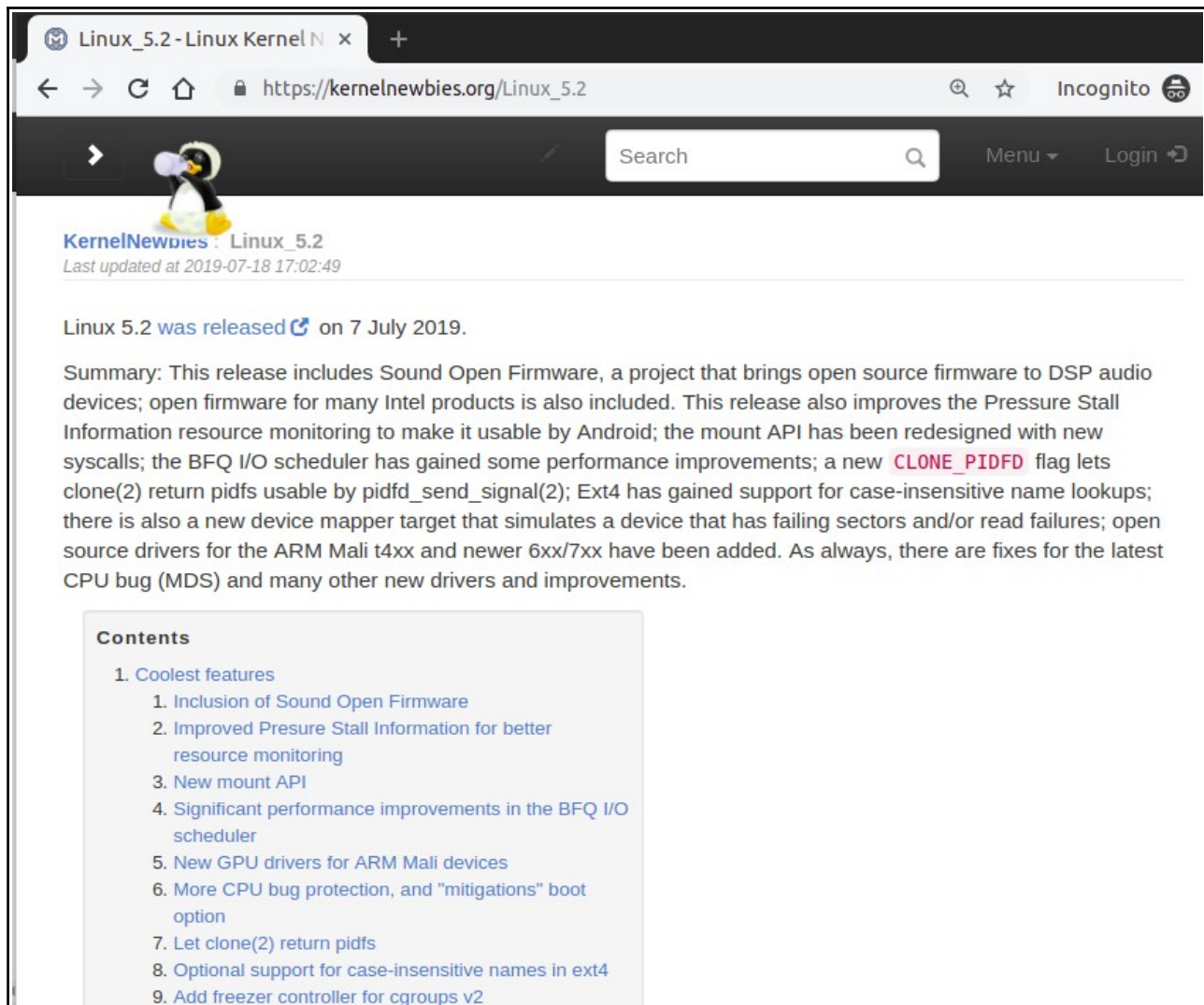
LWN runs kernel-oriented content every week. Some of our more recent articles in this area include:

March 29, 2018	An audit container ID proposal
March 27, 2018	Read-only dynamic data
March 26, 2018	Some 4.16 and -stable development statistics
March 22, 2018	Energy-aware scheduling on asymmetric systems
March 21, 2018	A "runtime guard" for the kernel

and the
kernelnewbies “Linux Changes” website !

1. The page
<http://kernelnewbies.org/LinuxChanges>

will have the *latest mainline kernel* changes information:
 << 5.2 at the time of this insertion (July 2019) >>



[SIDEBAR :: Get the Linux kernel 'finger banner']

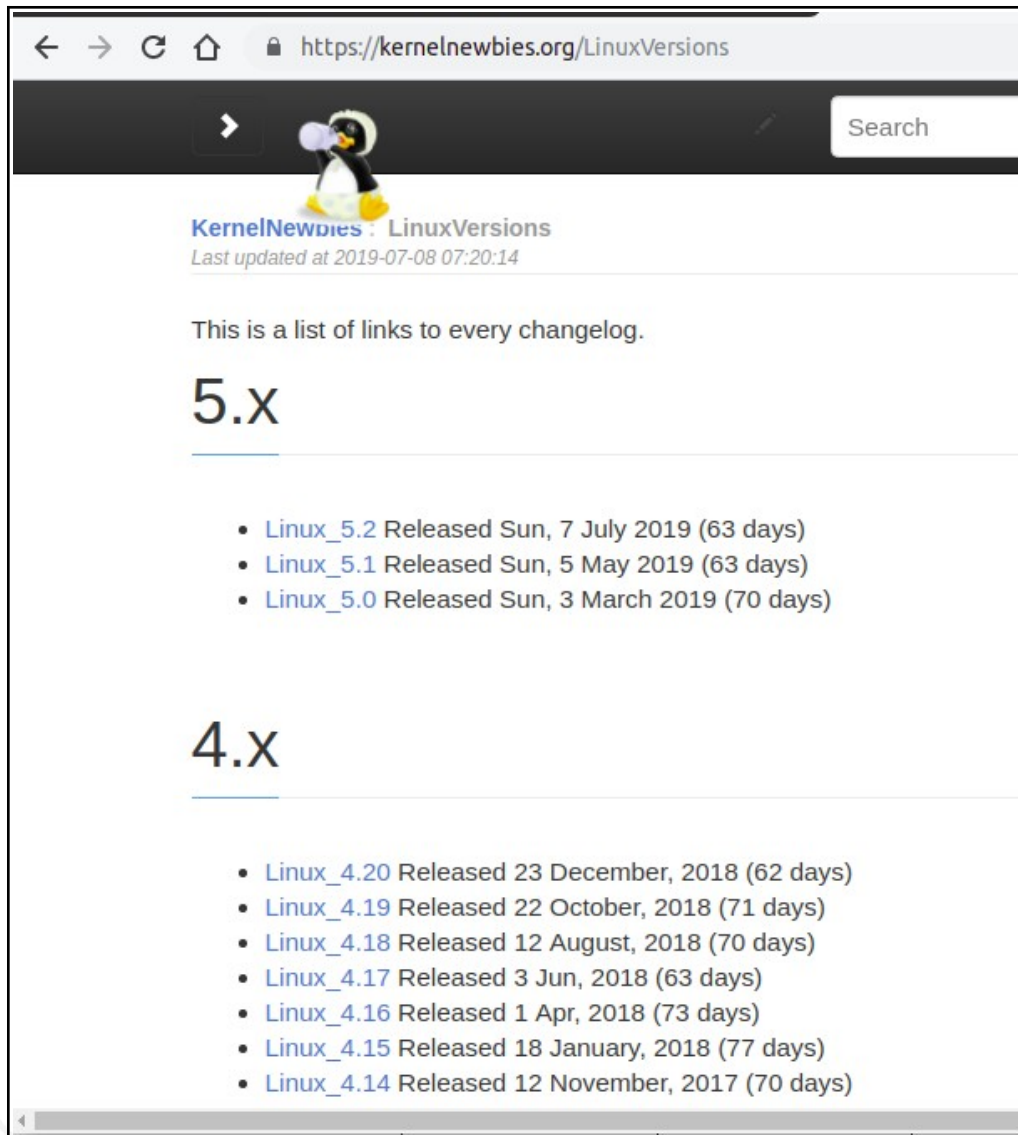
```
curl -L https://www.kernel.org/finger_banner
```

As of 01 Jan 2020

```
$ curl -L https://www.kernel.org/finger_banner
```

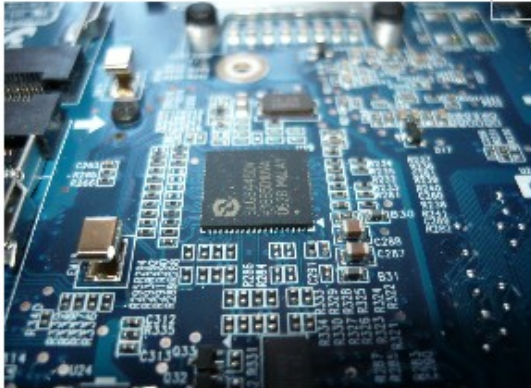
```
The latest stable version of the Linux kernel is: 5.4.7
The latest mainline version of the Linux kernel is: 5.5-rc4
The latest stable 5.4 version of the Linux kernel is: 5.4.7
The latest stable 5.3 version of the Linux kernel is: 5.3.18 (EOL)
The latest longterm 4.19 version of the Linux kernel is: 4.19.92
The latest longterm 4.14 version of the Linux kernel is: 4.14.161
The latest longterm 4.9 version of the Linux kernel is: 4.9.207
The latest longterm 4.4 version of the Linux kernel is: 4.4.207
The latest longterm 3.16 version of the Linux kernel is: 3.16.80
The latest linux-next version of the Linux kernel is: next-20191220
$
```


2. To see links to all kernel versions, goto
<http://kernelnewbies.org/LinuxVersions>



How would a professional Linux product company select a kernel version and what would the product life cycle be like? [See this Wikipedia content on RedHat's product life cycle and kernel backporting.](#)

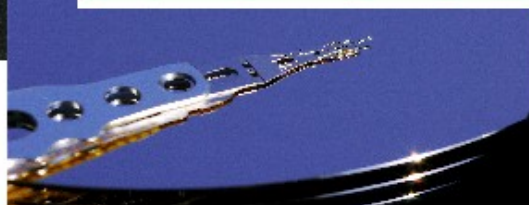
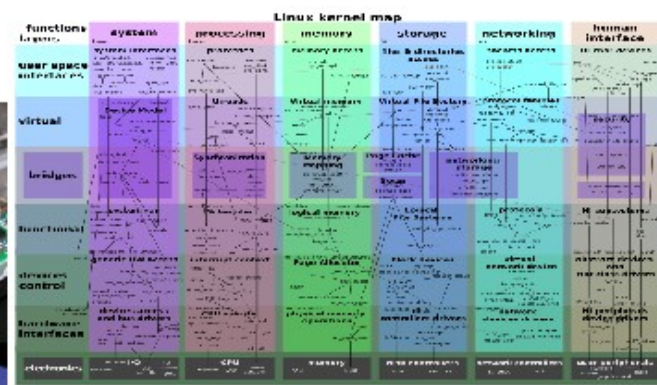
Linux Operating System Specialized



The highest quality Training on:

Linux Fundamentals, CLI and Scripting
Linux Systems Programming
Linux Kernel Internals
Linux Device Drivers
Embedded Linux
Linux Debugging Techniques
New! *Linux OS for Technical Managers*

Please do visit our website for details:
<http://kaiwantech.in>



<http://kaiwantech.in>

kaiwanTECH Linux OS Corporate Training Programs

Please do check out our current offering of world-class, seriously-valuable, high on returns, technical Linux OS corporate training programs here: <http://bit.ly/ktcorp>