



Sistemas Operativos Fisop 2022

El Kernel

La Perspectiva del Usuario



¿Qué vé el usuario?

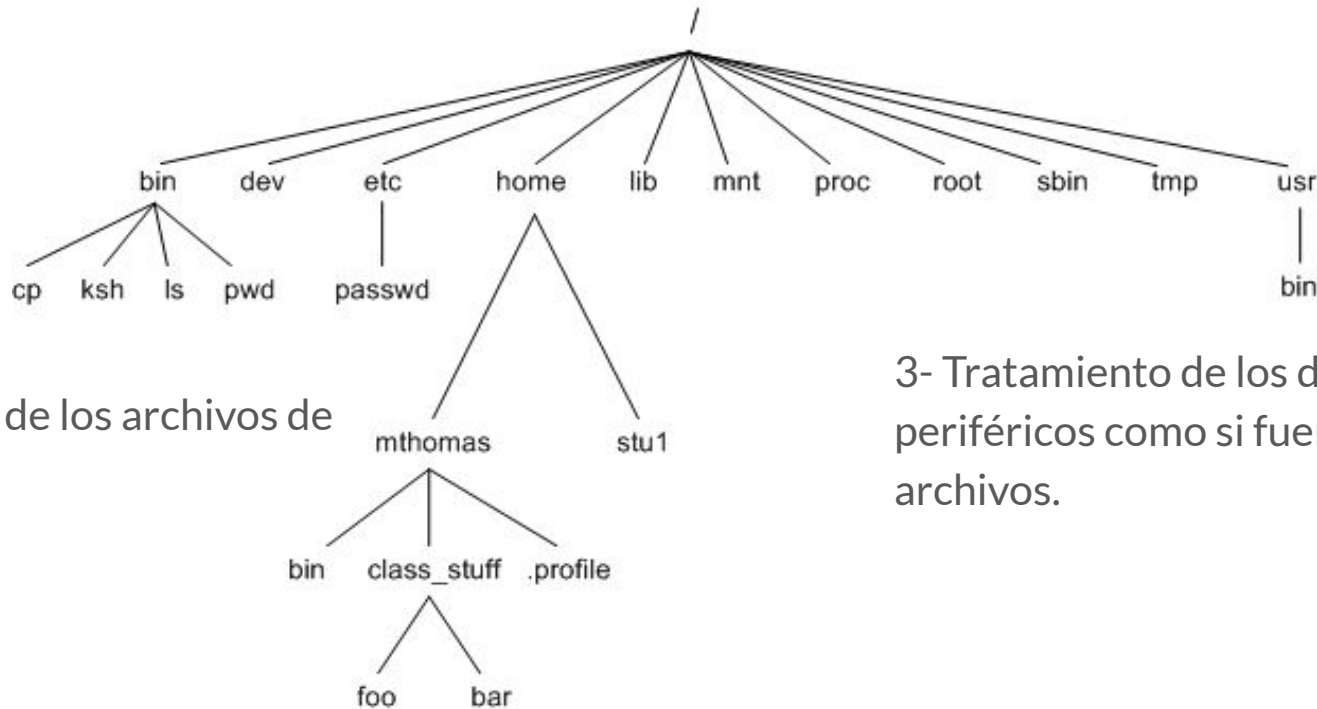
Desde el punto de vista del usuario lo que “se ve” es:

- **El Sistema de Archivos o File System.**
- **El Entorno de Procesamiento.**
- **Los Bloques Primitivos de Construcción.**

El Sistema de Archivos

1- estructura jerárquica

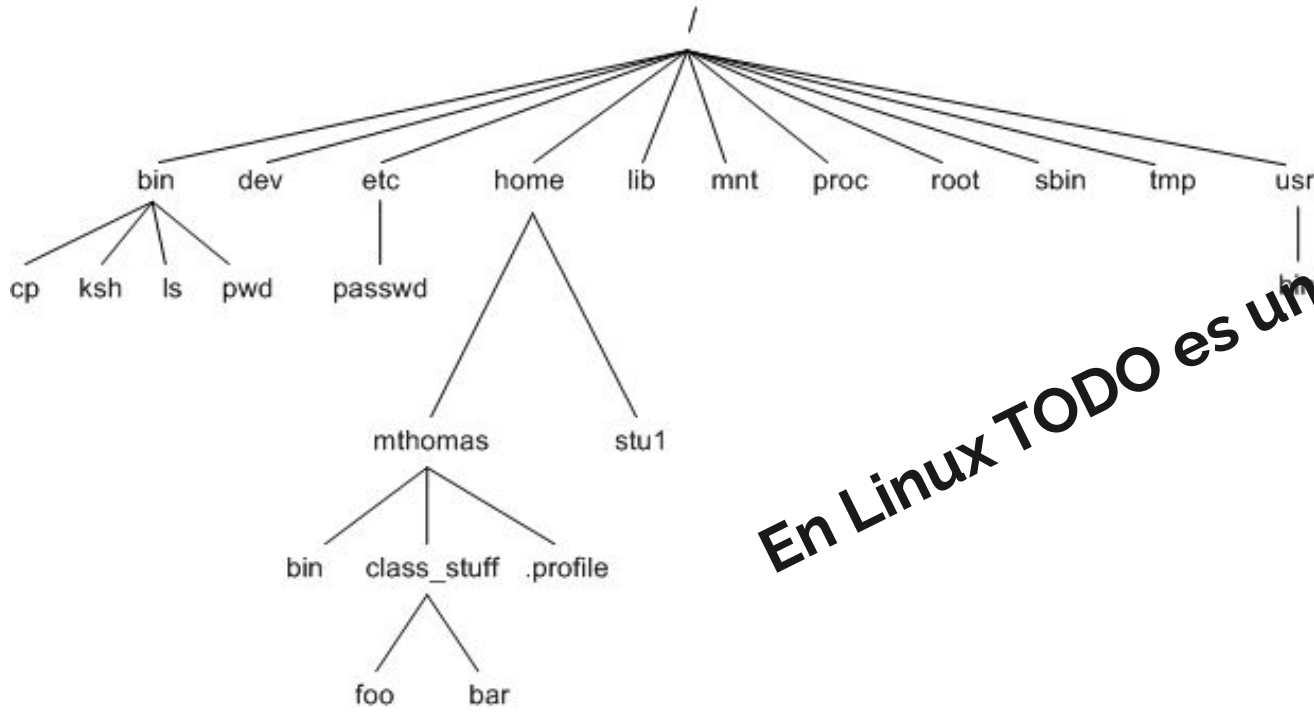
2- habilidad de crear y borrar archivos



4- protección de los archivos de datos

3- Tratamiento de los dispositivos periféricos como si fueran archivos.

El Sistema de Archivos



En Linux TODO es un archivo!!!!

El Entorno de Procesamiento

El intérprete de comandos o shell es el primer programa que se ejecuta (en un unix) en modo usuario, este a su vez ejecuta el comando de login.

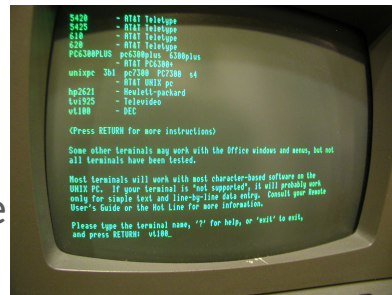
El intérprete de comandos captura las teclas que son presionadas en el teclado y las traduce a nombres de comandos.

El shell puede interpretar tres tipos de comandos:

- Comandos Ejecutables
- simples programas , por ejemplo el ls

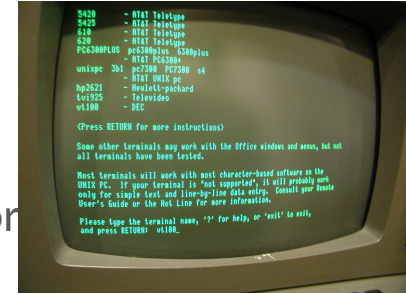
- Shell scripts

- Estructuras de control del Shell if-then-else-fi



El Entorno de Procesamiento

- El shell ejecuta los comandos buscando en ciertos directorios una determinada secuencia.
- Dado que el shell es un programa no forma parte del Kernel del sistema operativo
- El sistema operativo provee también unas System Calls para la creación y manipulación de procesos:
 - **fork()**: crea un proceso hijo a partir de un proceso padre.
 - **exec()**, **execve()**: creación de procesos.
 - **wait()**: sincronización



Bloques Primitivos de Construcción



- La idea de unix es la de proveer ciertos mecanismos para que los programadores construyan a partir de pequeños programas otros más complejos.

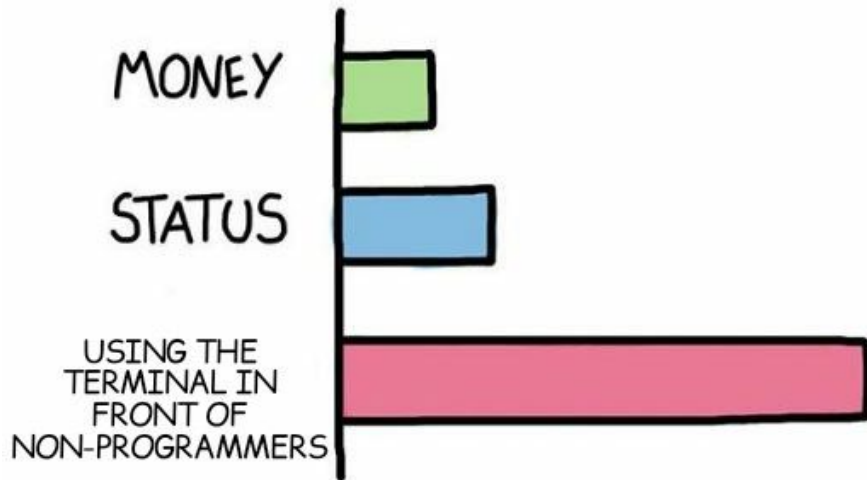
El **redireccionamiento de entrada y salida** de datos.

El operador **>** **redirecciona el standar output a un archivo.**

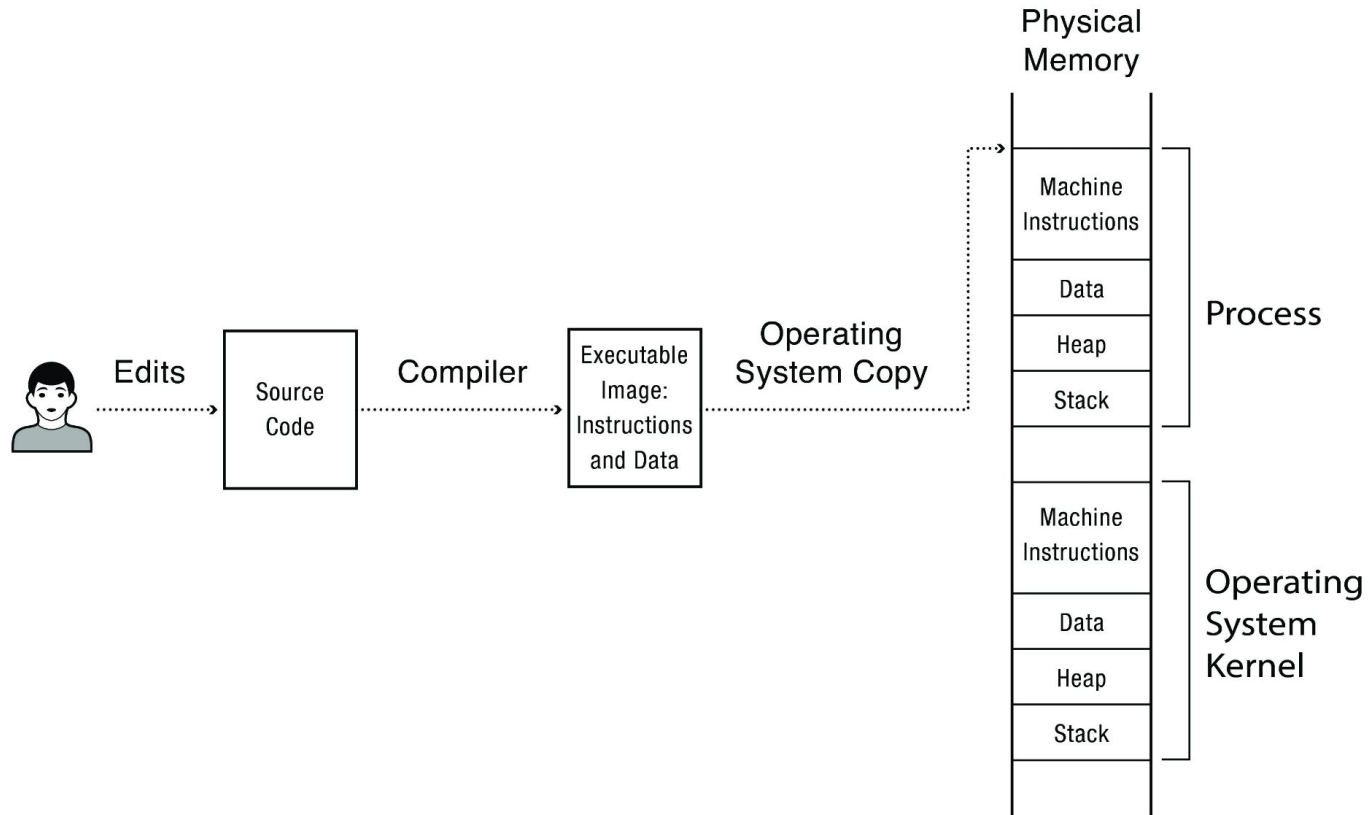
El operador **<** **redirecciona el standar input tomando los datos de un archivo.**

Las tuberías o **los pipes.**

WHAT GIVES PEOPLE FEELINGS OF POWER

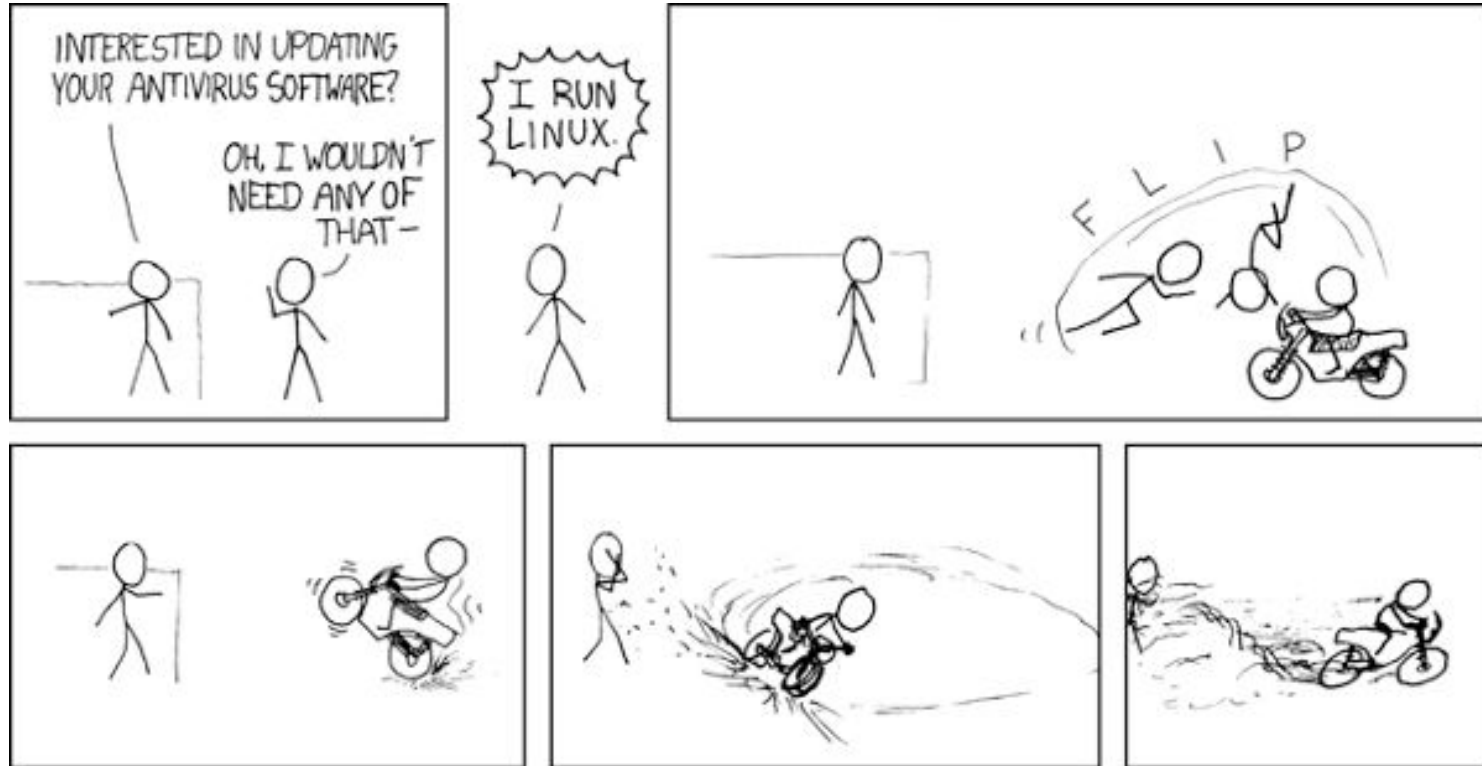


El kernel



De Programa a Proceso

¿Por Qué es Necesario el Kernel?





**El rol central de un sistema operativo es la
protección**

Ejecución Directa

OS

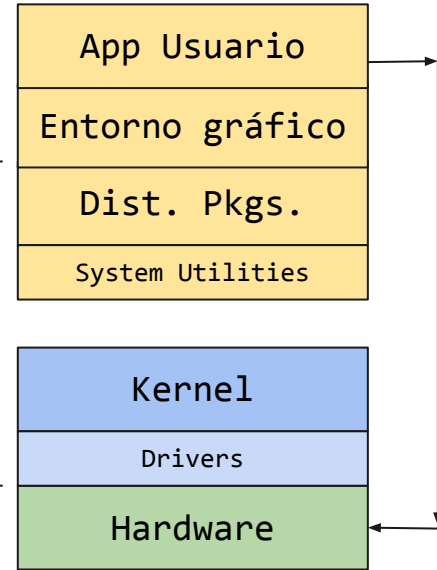
Create entry for process list
Allocate memory for program
Load program into memory
Set up stack with argc/argv
Clear registers
Execute **call** main()

Free memory of process
Remove from process list

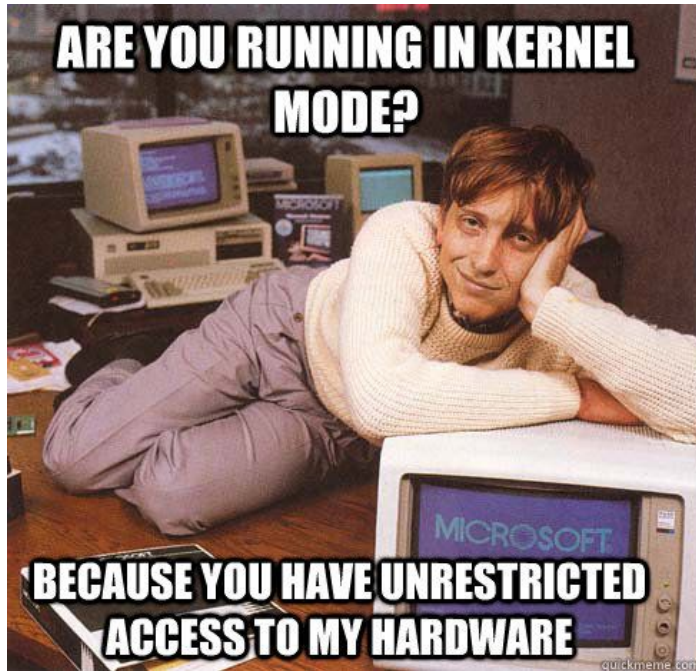
Program

Run main()
Execute **return** from main

Sistema Operativo



Ejecución Directa



This program requires Microsoft Windows.

C:\DOS>cd..

C:\>dir/w

Volume in drive C is MS-DOS_6
Volume Serial Number is 1B35-7518
Directory of C:\

[DOS]	COMMAND.COM	WINA20.386	CONFIG.SYS	PLAN2.COM
	5 file(s)	72,343 bytes		
		100,579,328 bytes free		

C:\>edit

C:\>test.bat

Bad command or file name
Bad command or file name
Bad command or file name
Bad command or file name
Bad command or file name
Bad command or file name
Bad command or file name

Ejecución Directa

C:\>dir/w

Volume in drive C is MS-DOS_6

Volume Serial Number is 462F-A26E

Directory of C:\

[DOS]	COMMAND.COM	WINA20.386	CONFIG.SYS	AUTOEXEC.BAT
PINGPONG.COM				
6 file(s)	66,352 bytes			
	100,712,448 bytes free			

C:\>pl_

Ejecución Directa

OS @ boot (kernel mode)

initialize trap table

Hardware

remember address of...
syscall handler

OS @ run (kernel mode)

Create entry for process list
Allocate memory for program
Load program into memory
Setup user stack with argv
Fill kernel stack with reg/PC
return-from-trap

Hardware

restore regs
(from kernel stack)
move to user mode
jump to main

Program (user mode)

Run main()
...
Call system call
trap into OS

Handle trap
Do work of syscall
return-from-trap

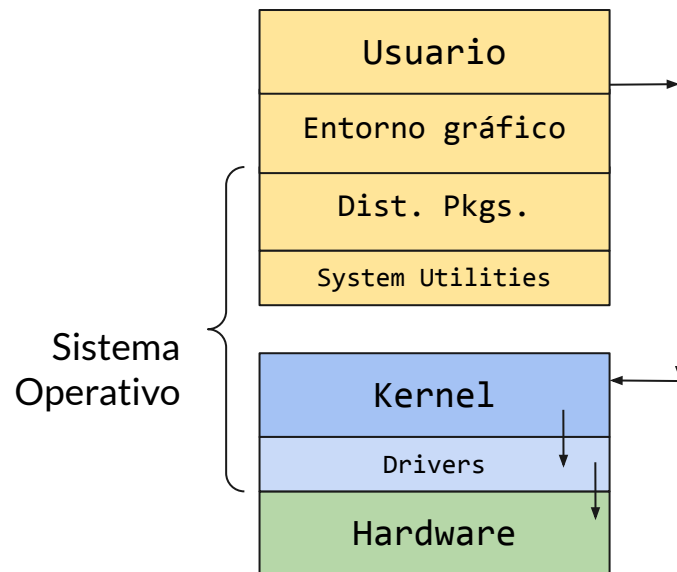
save regs
(to kernel stack)
move to kernel mode
jump to trap handler

restore regs
(from kernel stack)
move to user mode
jump to PC after trap

Free memory of process
Remove from process list

...
return from main
trap (via `exit()`)

Limitar la Ejecución Directa



Limitar la Ejecución Directa

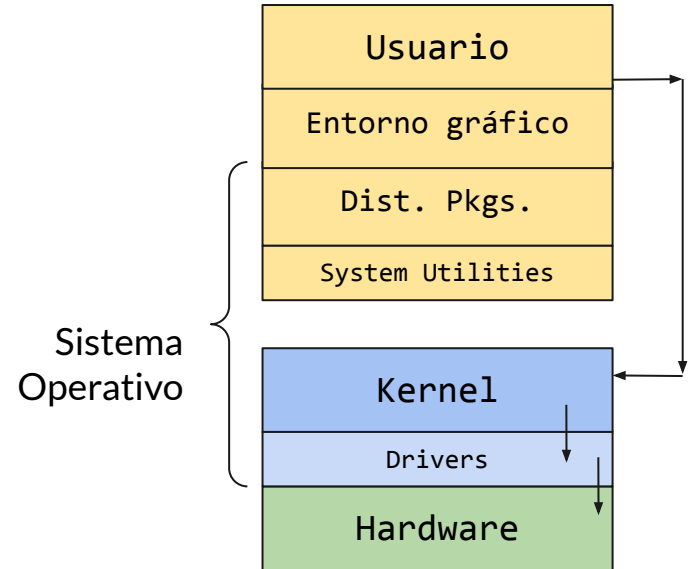
La ejecución directa es PELIGROSA!

Modo Dual.

Inst Privilegiadas.

Protección de Memoria.

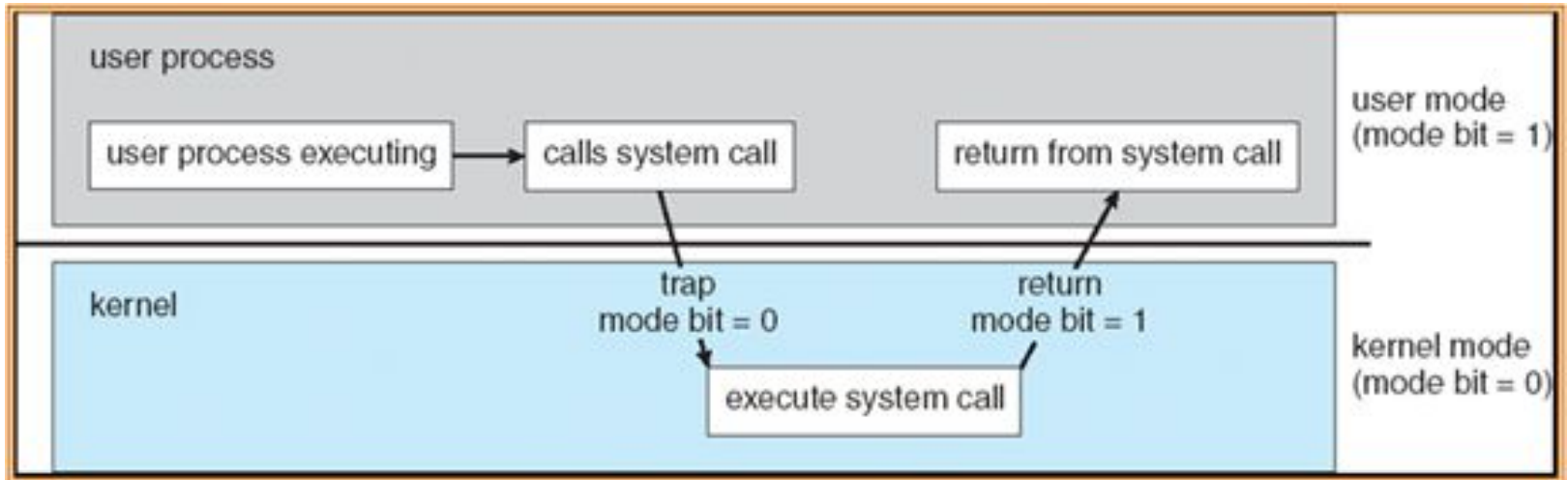
Timer Interrupts



Modo Dual de Operaciones

User-Mode

Kernel-Mode



IOPL

I/O Privilege level

Ocupa los bits 12 y 13 en
el registro FLAGS.

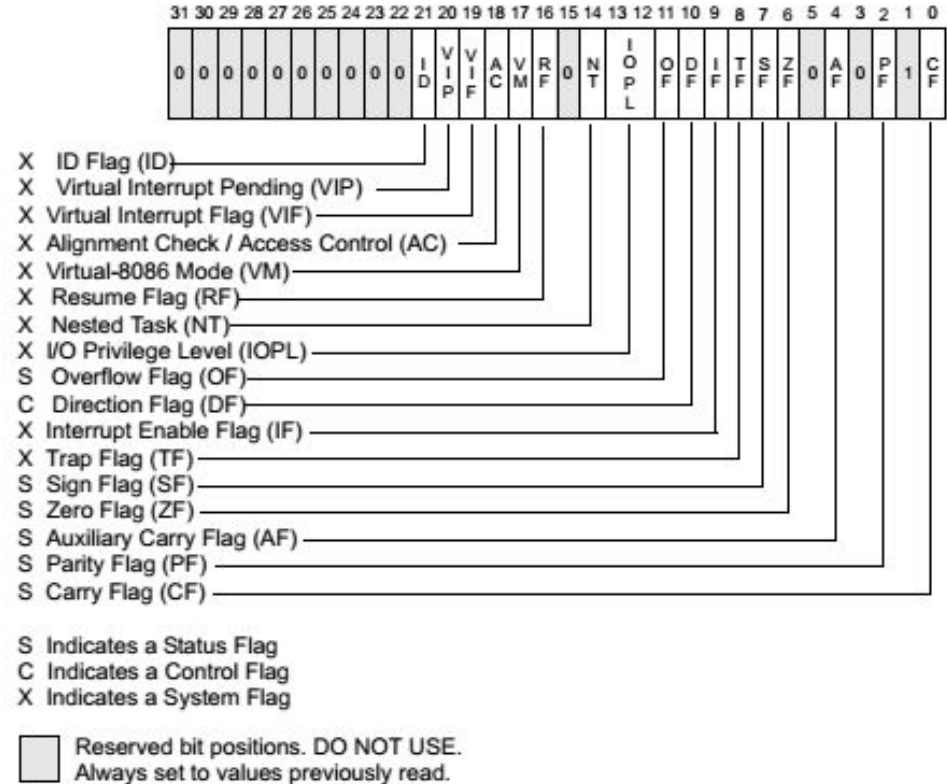
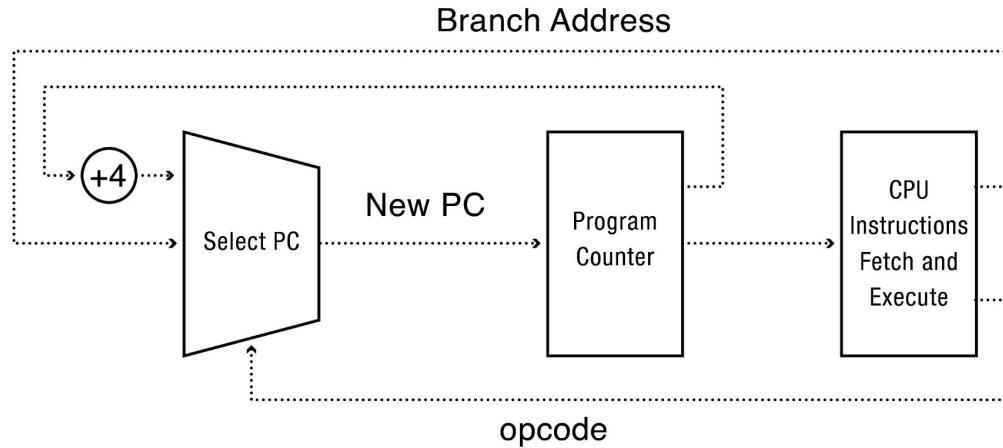


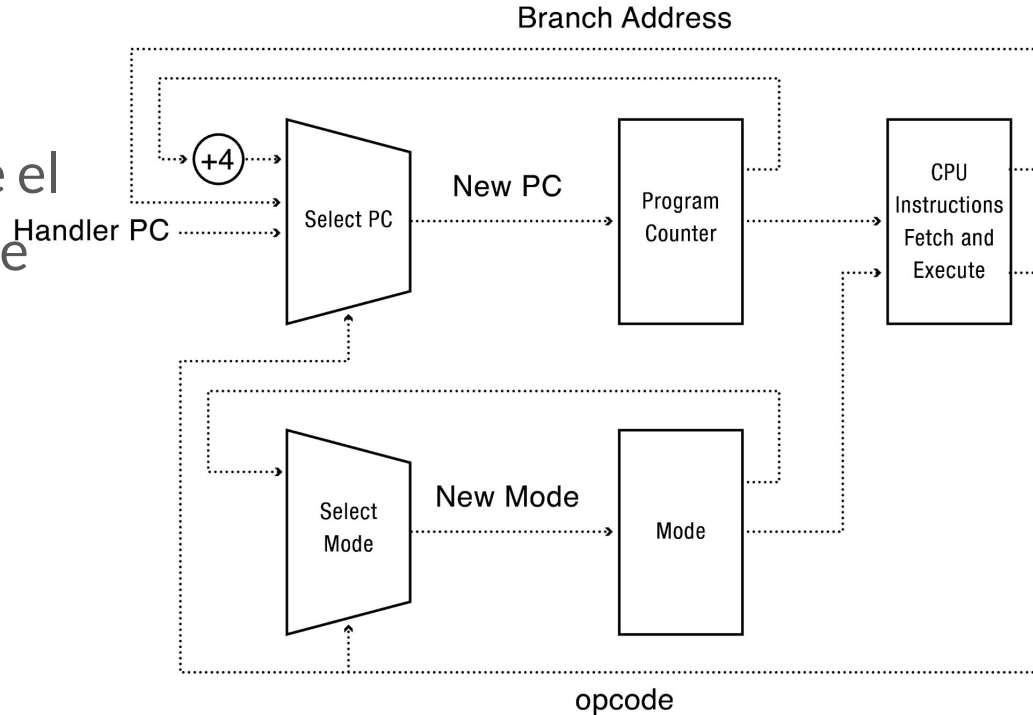
Figure 3-8. EFLAGS Register

Ejecución Directa



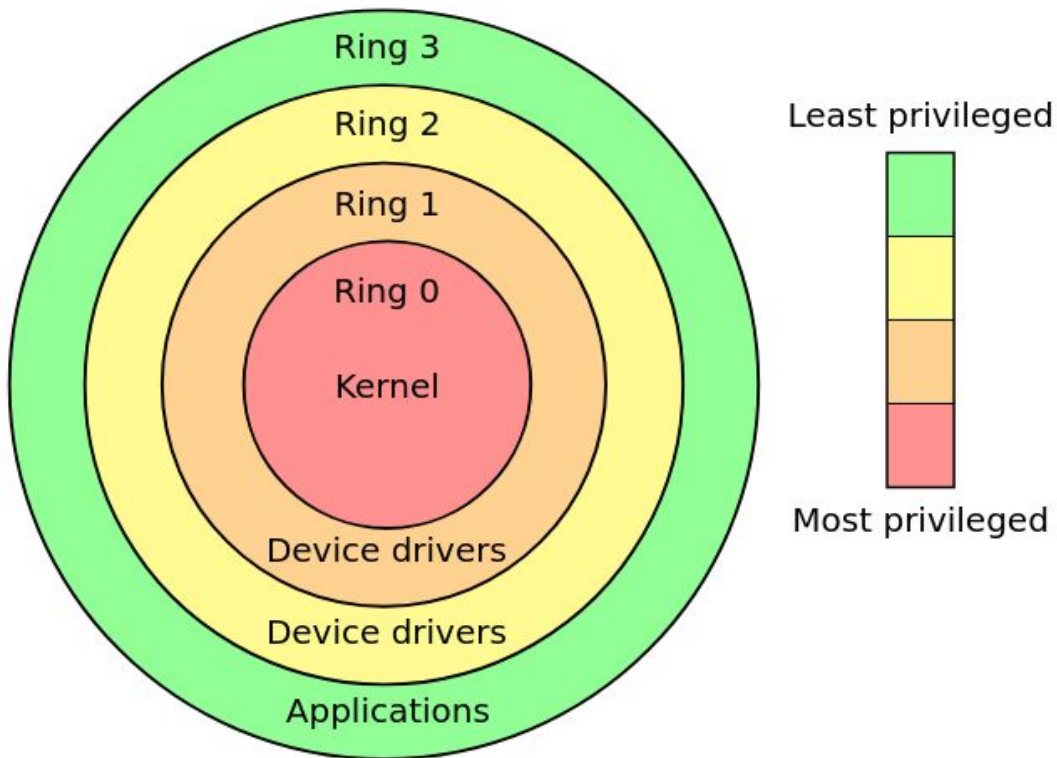
Modo Dual de Operaciones

Existen ciertos estados en el que el procesador puede encontrarse.



Kernel Land y User Land

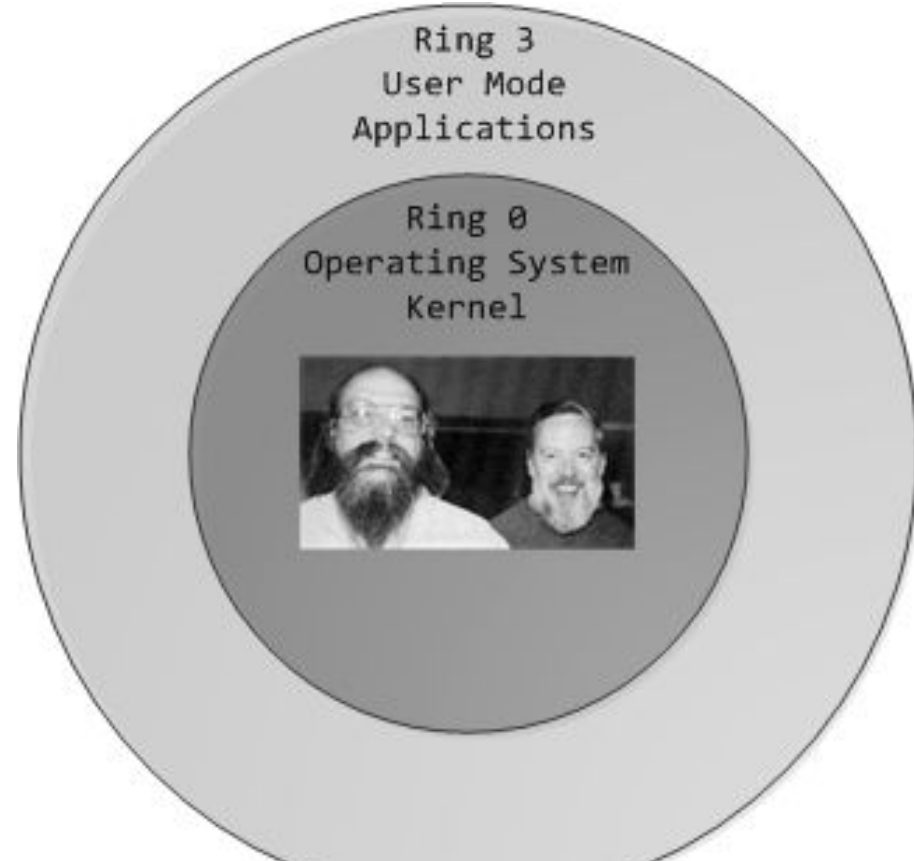
Cada ring puede ejecutar una determinada cantidad de instrucciones. Esto se detecta por hardware cada vez que una instrucción se ejecuta.



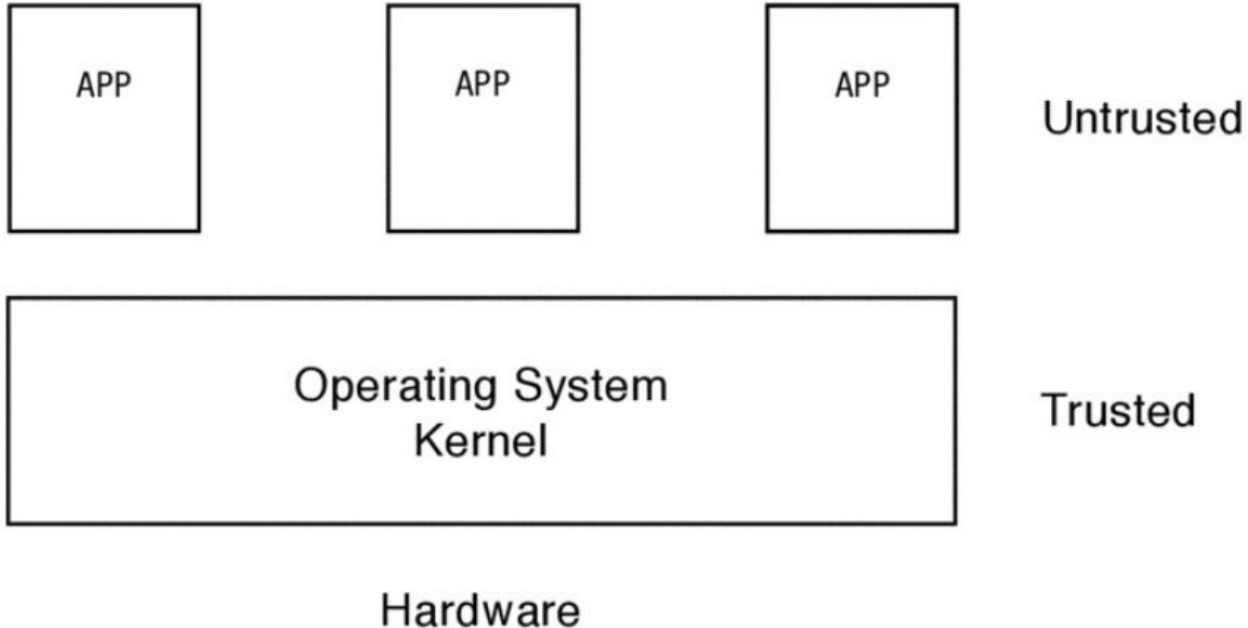
Kernerl Land y User Land

El kernel debe garantizar:

- Multiplexacion
- Aislamiento
- interaccion



Kernerl Land y User Land



Cómo protege el kernel de un SO las aplicaciones y los usuarios

- Instrucciones Privilegiadas
- Protección de Memoria
- Timer Interrupts



Instrucciones Privilegiadas

Distintos set de instrucciones



Instrucciones Privilegiadas



- LGDT — Load GDT register.
- LLDT — Load LDT register.
- LTR — Load task register.
- LIDT — Load IDT register.
- MOV (control registers) — Load and store control registers.
- LMSW — Load machine status word.
- CLTS — Clear task-switched flag in register CR0.
- MOV (debug registers) — Load and store debug registers.
- INVD — Invalidate cache, without writeback.
- WBINVD — Invalidate cache, with writeback.
- INVLPG — Invalidate TLB entry.
- HLT— Halt processor.
- RDMSR — Read Model-Specific Registers.
- WRMSR — Write Model-Specific Registers.
- RDPMC — Read Performance-Monitoring Counter.
- RDTSC — Read Time-Stamp Counter.

Instrucciones No Privilegiadas

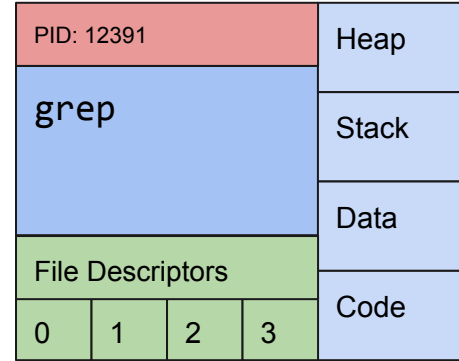
- Reading the status of Processor
- Reading the System Time
- Generate any Trap Instruction
- Sending the final printout of Printer



Protección de Memoria

Dirección Virtual vs Dirección Física

Espacio de direcciones



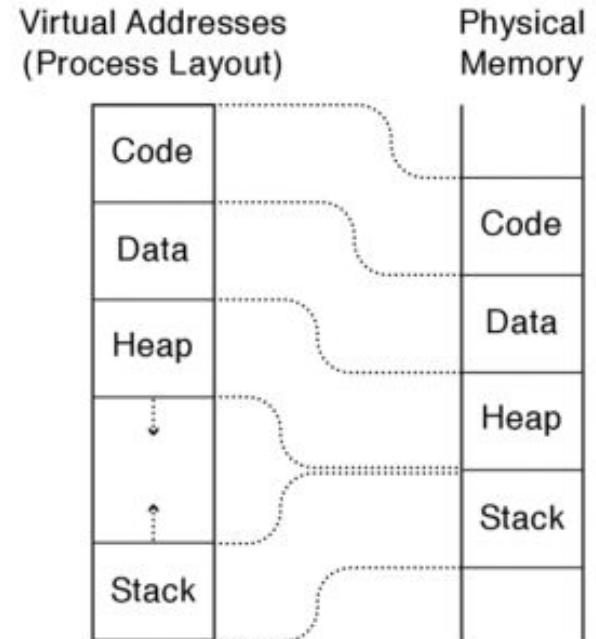
stdin
stdout
stderr
file.txt



Protección de Memoria

Dirección Virtual vs Dirección Física

Espacio de direcciones



Timer Interrupts (Interrupciones por temporizador)

¿Cómo hace el Kernel para volver a tener lo que un proceso tiene?

Casi todos los procesadores contienen un dispositivo llamado **Hardware Timer**, cada timer **interrumpe** a un determinado procesador mediante una interrupción por hardware.

Cuando una interrupción por tiempo se dispara se transfiere el control desde el proceso de usuario al Kernel



Timer Interrupts (Interrupciones por temporizador)



Veamoslo !

```
sudo cat /proc/timer_list
```



Modos de Transferencia

Modo de Transferencia

Una vez que el kernel pone a un proceso de usuario en un entorno aislado (sandbox), la próxima pregunta es: ¿cómo se transiciona entre un modo y otro?



Modo de Transferencia



Kernel Mode to User Mode

User Mode to Kernel Mode

User Mode to Kernel Mode



Interrupciones

Excepciones del Procesador

System Calls

Interrupciones

Una interrupción es una señal asincrónica hacia el procesador avisando que algún evento externo requiere su atención





Interrupciones

```
bob@susel:~> sudo cat /proc/interrupts
          CPU0
 0:         78   IO-APIC-edge      timer
 1:       9012   IO-APIC-edge      i8042
 3:          1   IO-APIC-edge
 4:       6609   IO-APIC-edge
 6:          7   IO-APIC-edge      floppy
 7:          0   IO-APIC-edge      parport0
 8:          1   IO-APIC-edge      rtc0
 9:          0   IO-APIC-fasteoi    acpi
12:     101705   IO-APIC-edge      i8042
14:       1374   IO-APIC-edge      ata_piix
15:      28050   IO-APIC-edge      ata_piix
16:        893   IO-APIC-fasteoi    Ensoniq AudioPCI
17:      77088   IO-APIC-fasteoi    ioc0, ehci_hcd:usb1
18:        112   IO-APIC-fasteoi    uhci_hcd:usb2
19:       5167   IO-APIC-fasteoi    eth0
40:          0   PCI-MSI-edge      PCIE PME, pciehp
41:          0   PCI-MSI-edge      PCIE PME, pciehp
42:          0   PCI-MSI-edge      PCIE PME, pciehp
43:          0   PCI-MSI-edge      PCIE PME, pciehp
44:          0   PCI-MSI-edge      PCIE PME, pciehp
45:          0   PCI-MSI-edge      PCIE PME, pciehp
46:          0   PCI-MSI-edge      PCIE PME, pciehp
```

Sudo cat /proc/interrupts

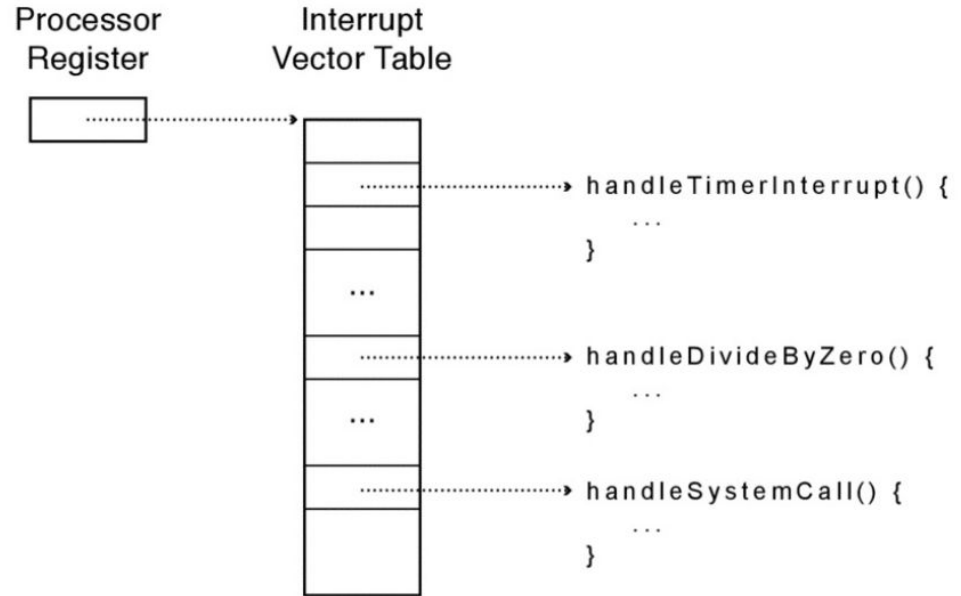
Orden de importancia

Errores de la Máquina
Timers
Discos
Network devices
Terminales
Interrupciones de Software

Interrupciones

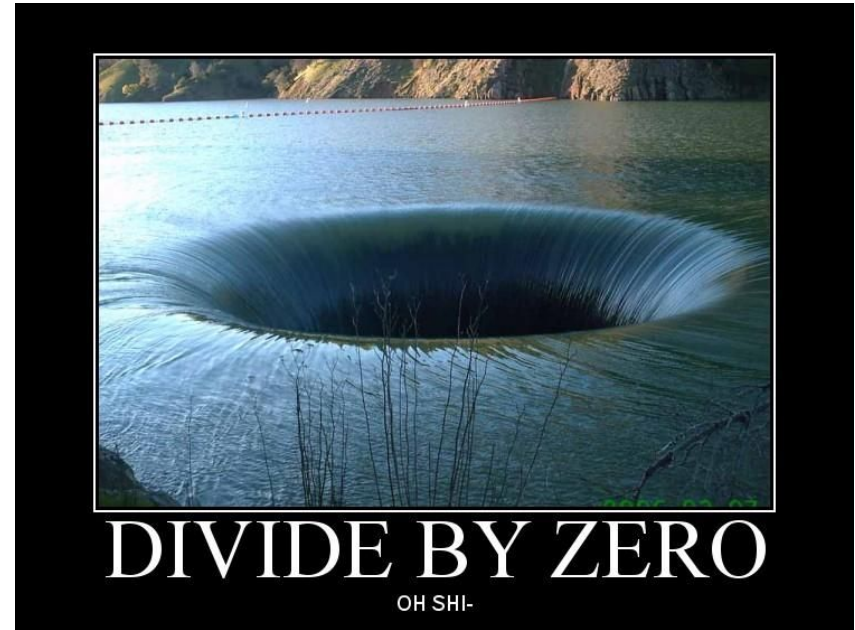
Orden de importancia

1. Errores de la Máquina
2. Timers
3. Discos Network devices
4. Terminales
5. Interrupciones de Software



Excepciones del Procesador

Una excepción es un evento de hardware causado por una aplicación de usuario que causa la transferencia del control al Kernel.



Excepciones del Procesador



Una excepción es un evento de hardware causado por una aplicación de usuario que causa la transferencia del control al Kernel.

```
set_trap_gate(0,&divide_error);
set_intr_gate(1,&debug);
set_intr_gate(2,&nmi);
set_system_intr_gate(3, &int3); /* int3/4 can be
called from all */
set_system_gate(4,&overflow);
set_trap_gate(5,&bounds);
set_trap_gate(6,&invalid_op);
set_trap_gate(7,&device_not_available);
set_task_gate(8,GDT_ENTRY_DOUBLEFAULT_TSS);
set_trap_gate(9,&coprocessor_segment_overrun);
set_trap_gate(10,&invalid_TSS);
set_trap_gate(11,&segment_not_present);
set_trap_gate(12,&stack_segment);
set_trap_gate(13,&general_protection);
set_intr_gate(14,&page_fault);
set_trap_gate(15,&spurious_interrupt_bug);
set_trap_gate(16,&coprocessor_error);
set_trap_gate(17,&alignment_check);
```

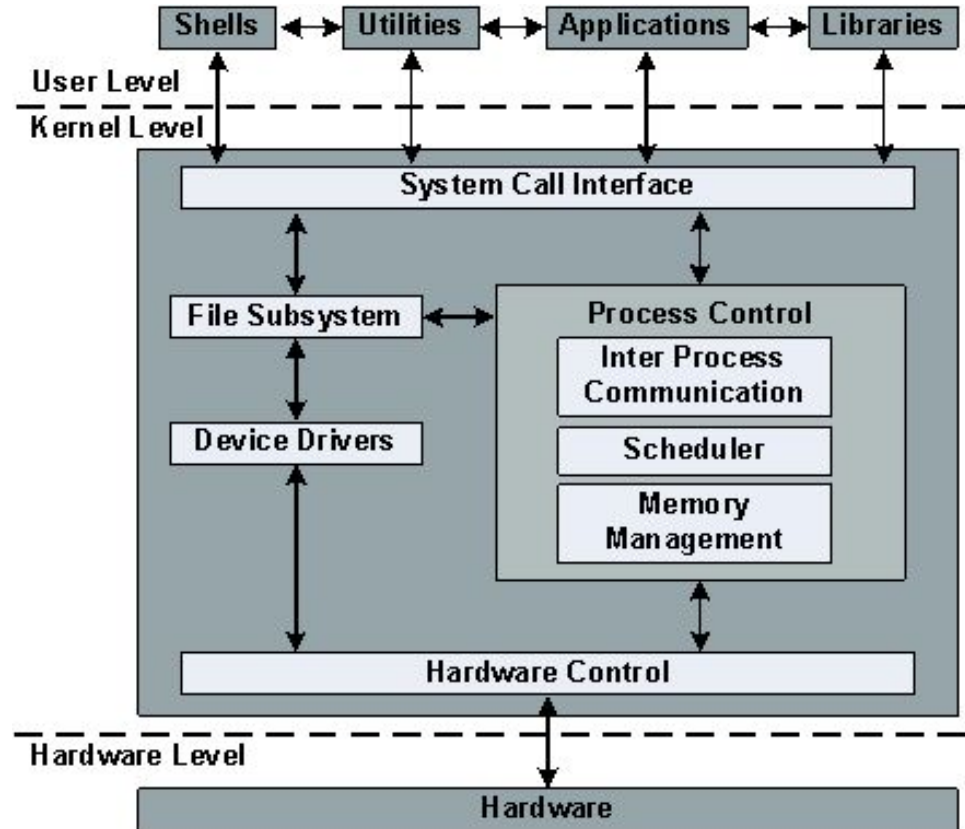
System Calls

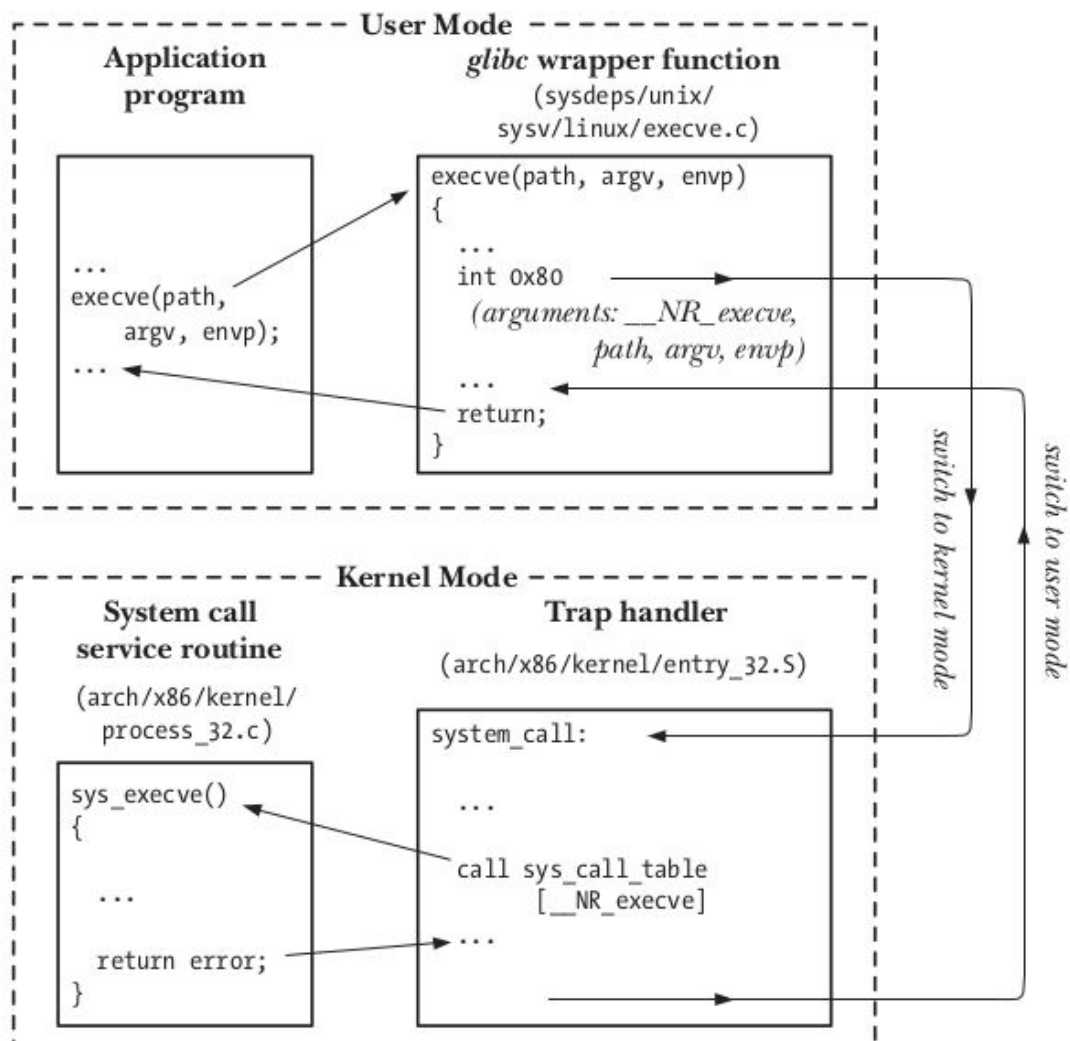
Un proceso de usuario puede hacer que la transición de modo sea hecha voluntariamente.

Una **System Call** es algún procedimiento provisto por el kernel que puede ser llamada desde el user level.



System Calls





System Calls



Desde el punto de vista de un programa llamar a una system call es más o menos como invocar a una función de C. Por supuesto, detrás de bambalinas muchas cosas suceden:

El programa realiza un llamado a una system call mediante la invocación de una **función wrapper** (envoltorio) en la biblioteca de C.

Dicha función wrapper tiene que proporcionar todos los argumentos al system call `trap_handling`. Estos argumentos son pasados al wrapper por el stack, pero el kernel los espera en determinados registros. La función wrapper copia estos valores a los registros.

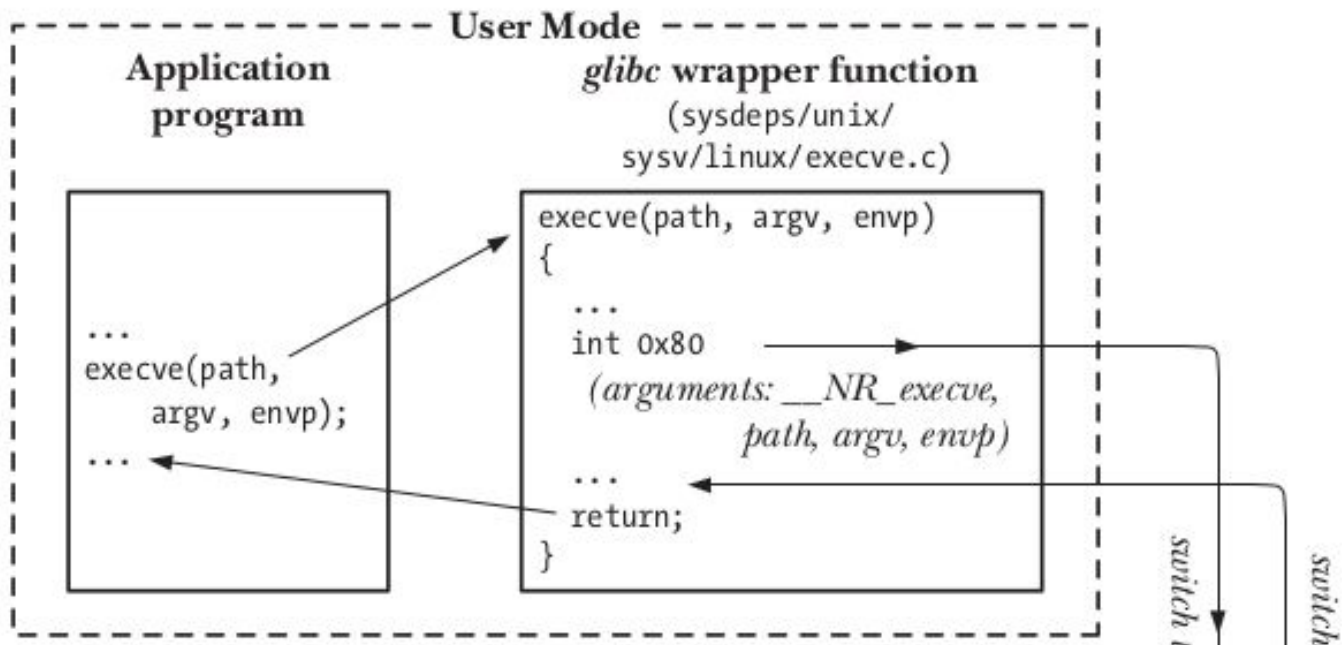
System Calls



Dado que todas las system calls son accedidas de la misma forma, el kernel tiene que saber identificarlas de alguna forma. Para poder hacer esto, la función wrapper copia el número de la system call a un determinado registro de la CPU (%eax).

La función wrapper ejecuta una instrucción de código máquina llamada trap machine instruction (int 0x80), esta causa que el procesador pase de user mode a kernel mode y ejecute el código apuntado por la dirección 0x80 (128) del vector de traps del sistema.

System Calls



System Calls



```
* Emulated IA32 system calls via int 0x80.  
*  
* Arguments:  
* %eax System call number.  
* %ebx Arg1  
* %ecx Arg2  
* %edx Arg3  
* %esi Arg4  
* %edi Arg5  
* %ebp Arg6 [note: not saved in the stack frame, should not be touched]  
*
```

System Calls

#	Name	Registers						Definition
		eax	ebx	ecx	edx	esi	edi	
0	sys_restart_syscall	0x00	-	-	-	-	-	kernel/signal.c:2475
1	sys_exit	0x01	int error_code	-	-	-	-	kernel/exit.c:935
2	sys_fork	0x02	-	-	-	-	-	kernel/fork.c:2116
3	sys_read	0x03	unsigned int fd	char __user *buf	size_t count	-	-	fs/read_write.c:566
4	sys_write	0x04	unsigned int fd	const char __user *buf	size_t count	-	-	fs/read_write.c:581
5	sys_open	0x05	const char __user *filename	int flags	umode_t mode	-	-	fs/fhandle.c:257
6	sys_close	0x06	unsigned int fd	-	-	-	-	fs/open.c:1153

<https://syscalls32.paolostivanin.com/>

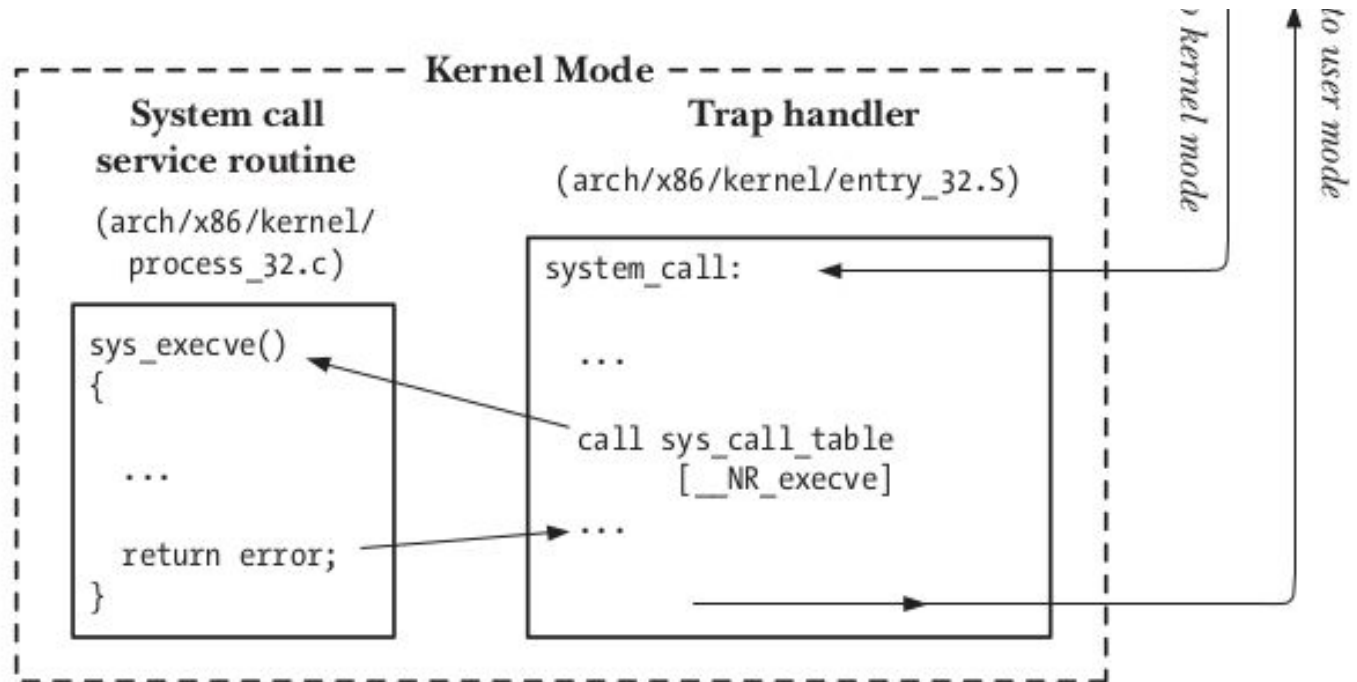
System Calls



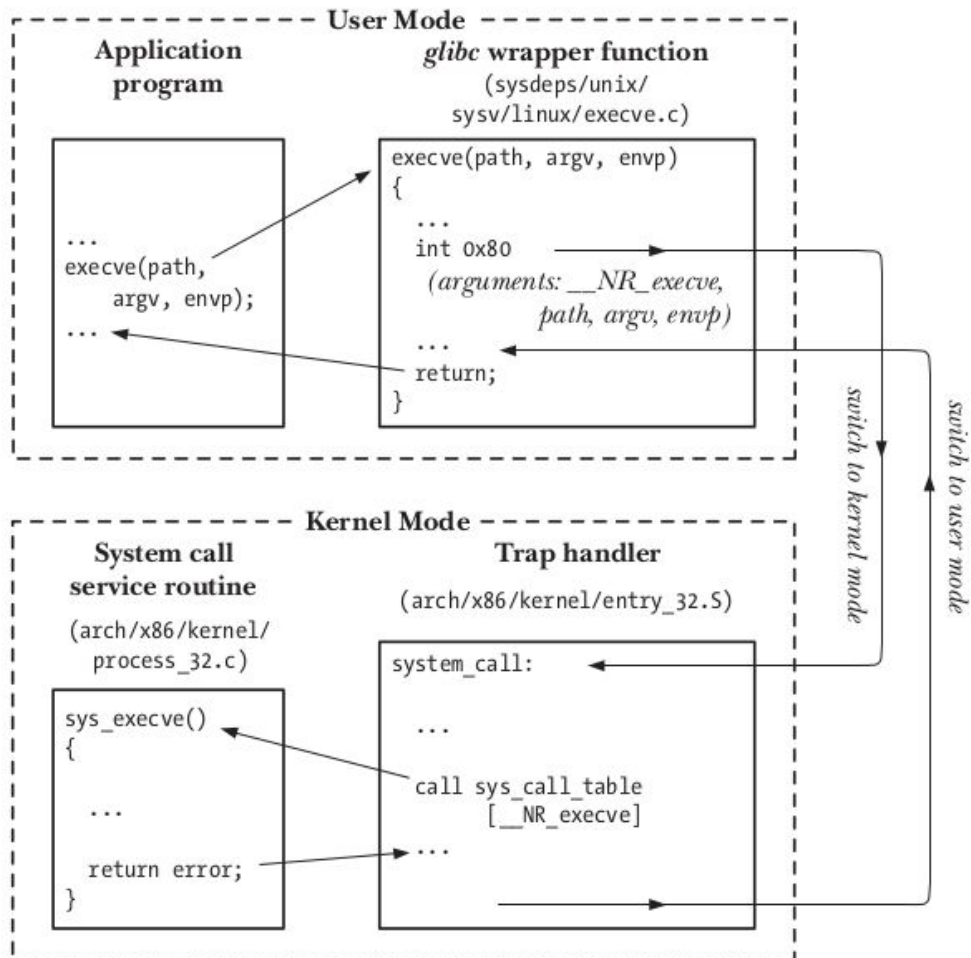
En respuesta al trap de la posición 128, el kernel invoca su propia función llamada `sys_call()` (`arch/i386/entry.s`) para manejar esa trap. Este manejador:

1. Graba el valor de los registros en el stack del kernel.
2. Verifica la validez del número de system call.
3. Invoca el servicio correspondiente a la system call llamada a través del vector de system calls, el servicio realiza su tarea y finalmente le devuelve un resultado de estado a la rutina `system_call()`.
4. Se restauran los registros almacenados en el stack del kernel y se agrega el valor de retorno en el stack.
5. Se devuelve el control al wrapper y simultáneamente se pasa a user mode.
6. Si el valor de retorno de la rutina de servicio de la system call da error, la función wrapper setea el valor en `errno`.

System Calls



System Calls



System Calls



Strace

Ltrace

```
strace ls testdir/
```

```
strace -o trace.log ls testdir/
```

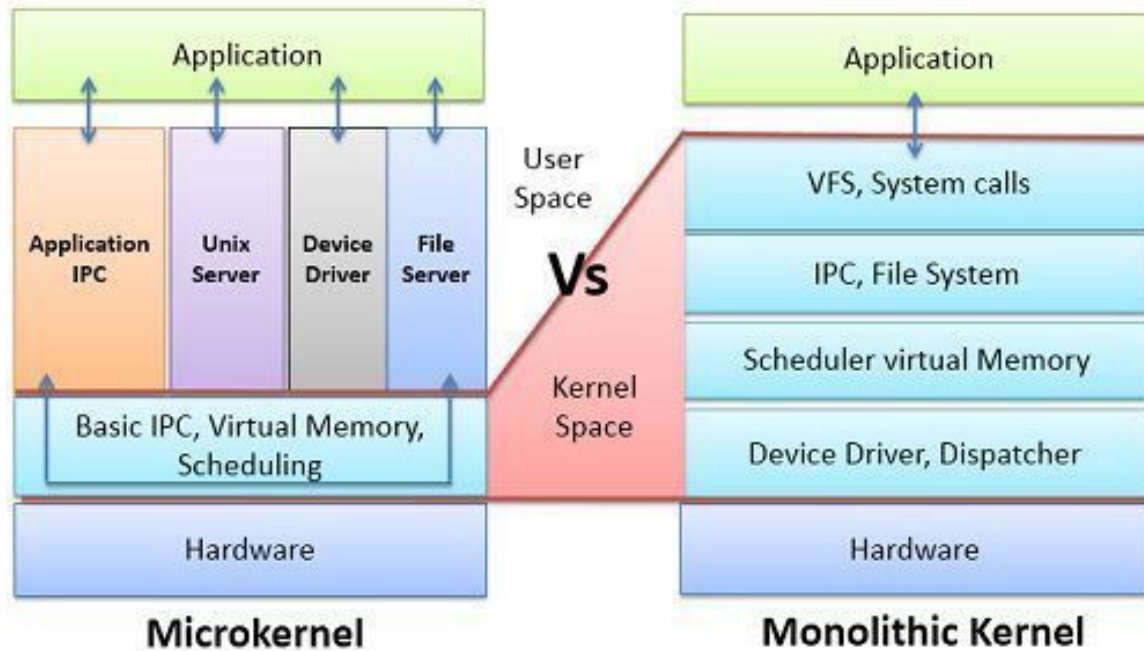
Kernel Mode to User Mode



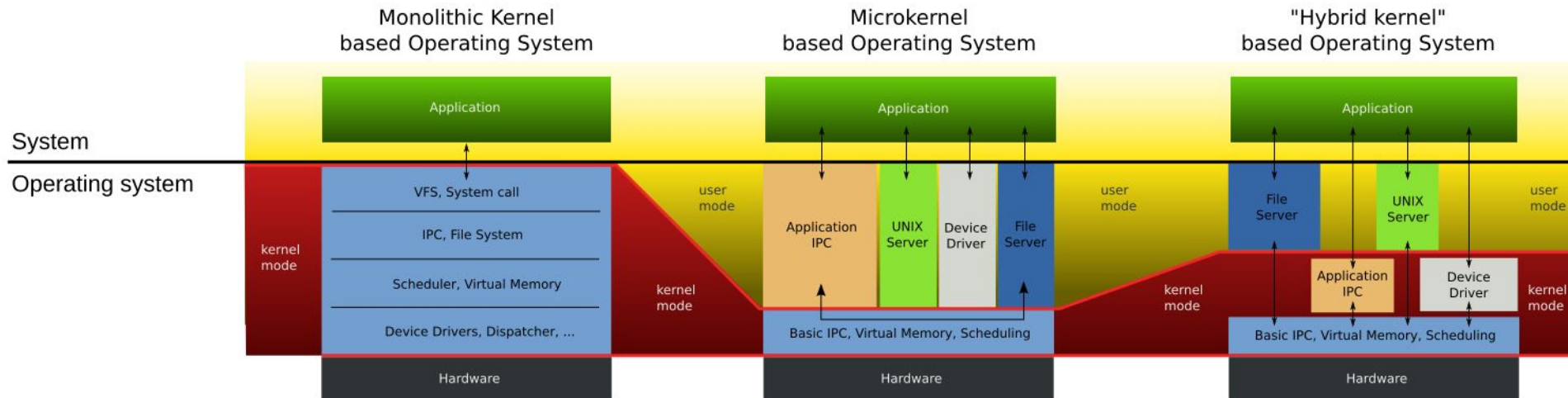
- Nuevo Proceso
- Continuar luego de una interrupción, excepción o una sys call
- Cambiar entre diferentes procesos

Dahlin Cap 2 hasta 2.4!

Tipos de Kernel



Tipos de Kernel



Tipos de Kernel

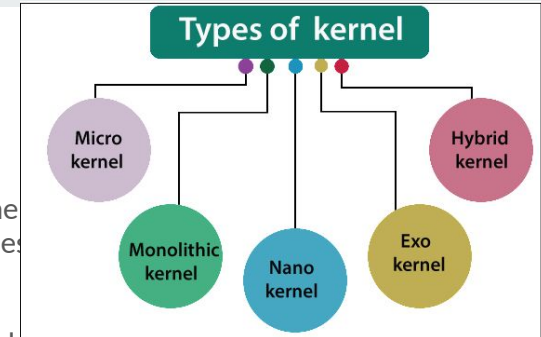
Monolithic Kernel: - In a monolithic kernel, the kernel and operating system, both run in the user space where security is not a major concern. The result of the monolithic kernel is fast access to hardware. Like if a device driver has a bug, then there may be chances of a whole system crash.

Microkernel: - A Microkernel is the derived version of the monolithic kernel. In microkernel, the kernel itself can do different jobs, and there is no requirement of an additional GUI.

Nano kernel: - Nano kernel is the small type of kernel which is responsible for hardware abstraction, but without system services. Nano kernel is used in those cases where most of the functions are set up outside.

Exo kernel: - Exo kernel is responsible for resource handling and process protection. It is used where you are testing out an inhouse project, and in up-gradation to an efficient kernel type.

Hybrid kernel: - Hybrid kernel is a mixture of microkernel and monolithic kernel. The Hybrid kernel is mostly used in Windows, Apple's macOS. Hybrid kernel moves out the driver and keeps the services of a system inside the kernel.





Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (0% complete)

If you'd like to know more, you can search online later for this error: HAL_INITIALIZATION_FAILED

Y si el Kernel Falla ...

A problem has been detected and Windows has been shut down to prevent damage to your computer.

UNMOUNTABLE_BOOT_VOLUME

If this is the first time you've seen this error screen, restart your computer. If this screen appears again, follow these steps:

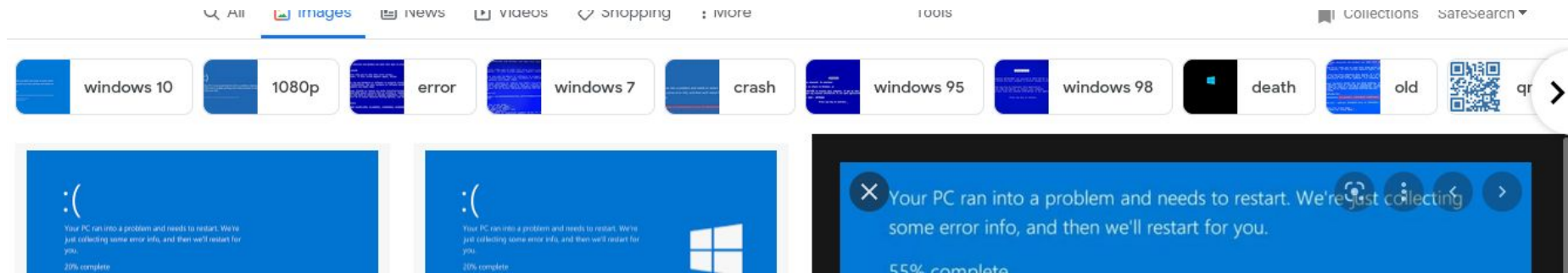
Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical Information:

*** STOP: 0x000000ED (0x80F128D0, 0xc000009c, 0x00000000, 0x00000000)

Y si el Kernel Falla ...



Y si el Kernel Falla ...

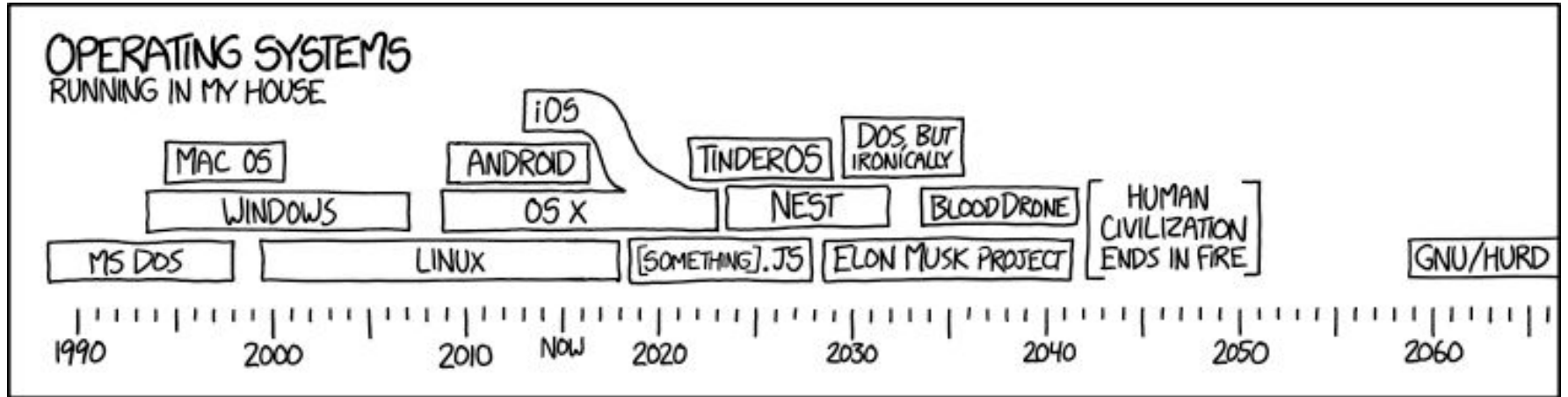

```
[ 0.077048] ACPI Error: Method parse/execution failed \_SB.PCI0._OSC, AE_ALRE  
ADY_EXISTS (20170531/psparse-550)  
[ 0.556042] tpm tpm0: A TPM error (?) occurred attempting to read a pcr value  
[ 0.668044] tpm tpm0: A TPM error (?) occurred attempting to read a pcr value  
[ 0.895361] Kernel panic - not syncing: VFS: Unable to mount root fs on unknow  
n-block(0,0)  
[ 0.895394] CPU: 1 PID: 1 Comm: swapper/0 Not tainted 4.13.0-generic #40~1
```

```
[ 97.131322] Call trace:  
00 00 00 00 00 00 00 00 00 00 00 00 00 00 ac c  
00 00 00 00 00 00 00 00 00 00 00 00 00 00 f  
c0 f7 74 70 c0 f7 40 10 80 f7 40 10 80 f  
[ 97.149858] EIP: [<f780000f>] 0xf780000  
[ 97.152388] ---[ end trace ca143223eef  
[ 97.153402] Kernel panic - not syncing
```

```
[ 0.000000] Kernel panic - not syncing: VFS: Unable to mount root fs  
on unknown-block(0,0)
```

Y si el Kernel Falla ...

Unix

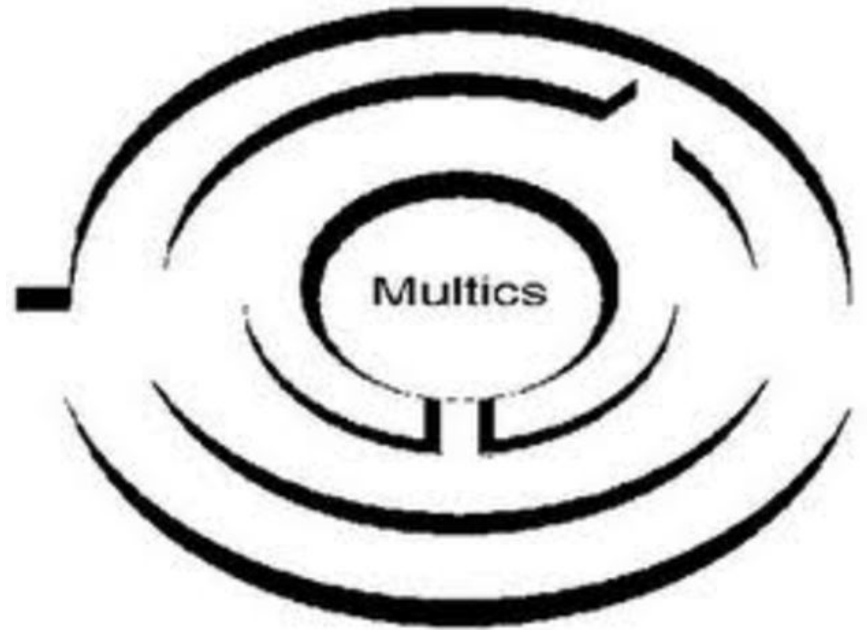


Multics



Multiplexed Information
and Computing Service

1964



Unix: Las Versiones Académicas



UNIX-v1 (Nov. 3, 1971): En su primera edición el sistema operativo ocupaba 16 Kb, 8Kb para programas de usuario, un disco de 512 Kb y un límite de 64 kb por archivo. Escrito en assembler.

UNIX-v2 (Jun. 12, 1972)

UNIX-v3 (Feb. 1973): Versión que tenía un compilador C.

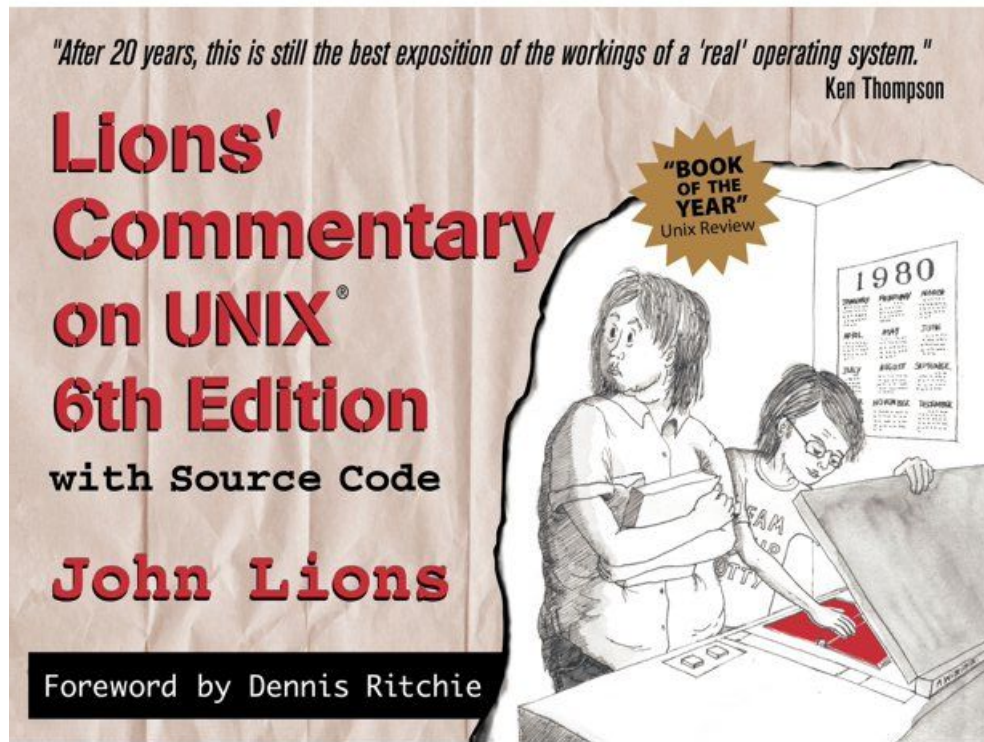
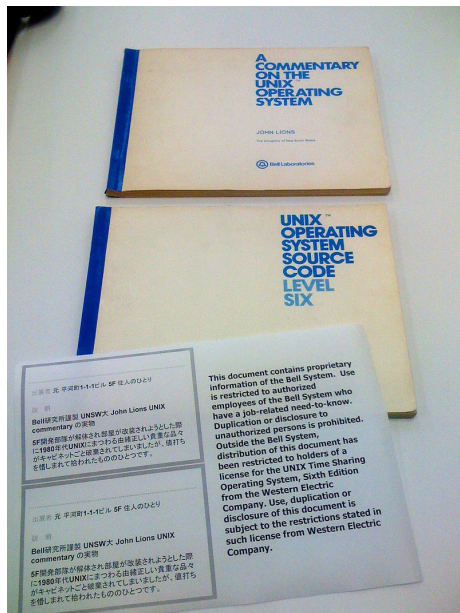
UNIX-v4 (Nov. 1973): Primera versión escrita totalmente en C.

UNIX-v5 (Jun. 1974)

UNIX-v6 (May 1975): Tal vez la versión más conocida por su licencia gratuita para investigación y enseñanza.

UNIX-v7 (Jan. 1979): Versión padre de todos los sistemas operativos basados en UNIX (excepto Coherent, Minix, and Linux).

Unix: V6



<https://warsus.github.io/lions-/>

Linux





Linus Benedict Torvalds



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 (torv...@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 :-).





Ubuntu



Fedora



Debian



Arch



Gentoo



Linux
From
Scratch



OS X



Windows



Iniciar al Sistema Operativo y el Kernel

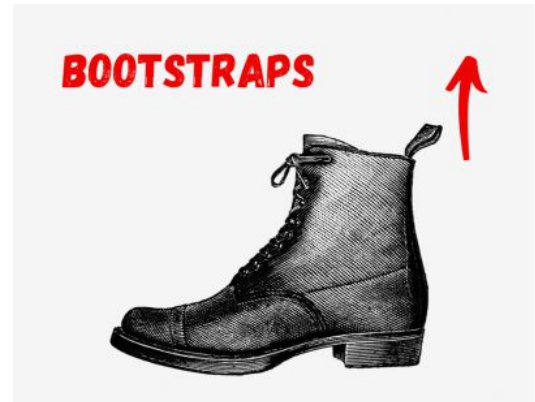
Inicio de S.O.



1. Booteo
2. Carga del Kernel
3. Inicio de las Aplicaciones de Usuarios

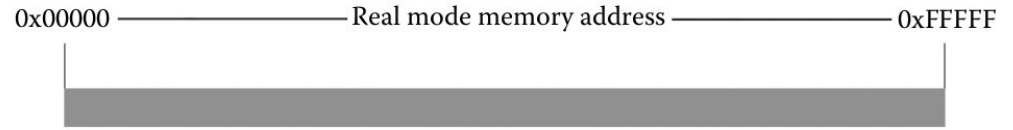
Booteo o Bootstrap

Este proceso es denominado bootstrap, y generalmente depende del hardware de la computadora. En él se realizan los chequeos de hardware y se carga el bootloader, que es el programa encargado de cargar el Kernel del Sistema Operativo. Este proceso consta de 4 partes.

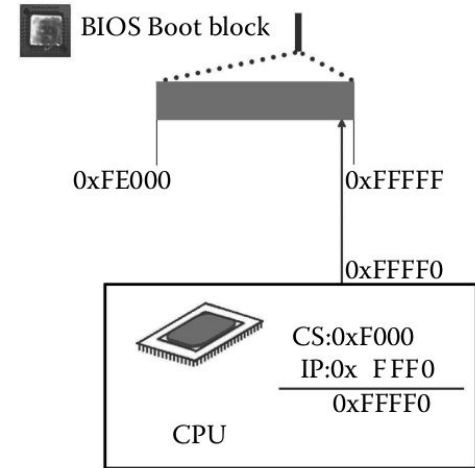


Booteo 1

Cargar el BIOS (Basic Input/Output System): para eso apenas se enciende la PC, se carga CS con 0xFFFF y IP con 0x0000; por ende, la dirección de CS:IP es 0xFFFF0, justamente la dirección de memoria de la BIOS.



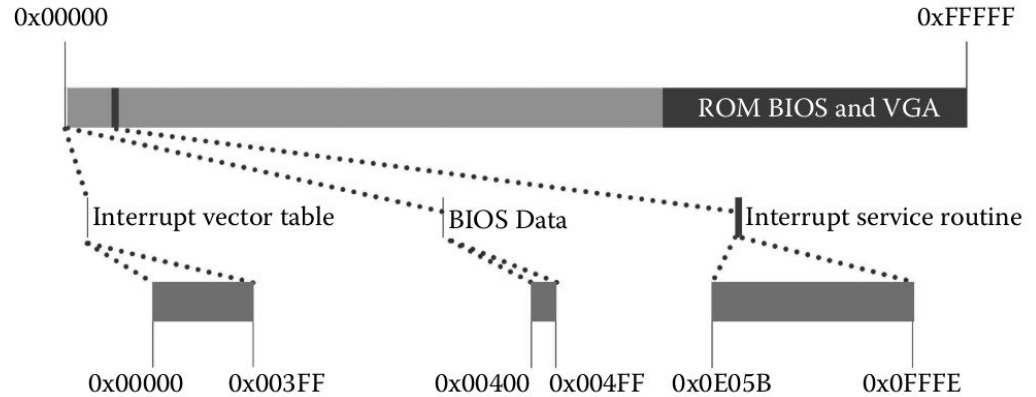
Power on



Booteo 2

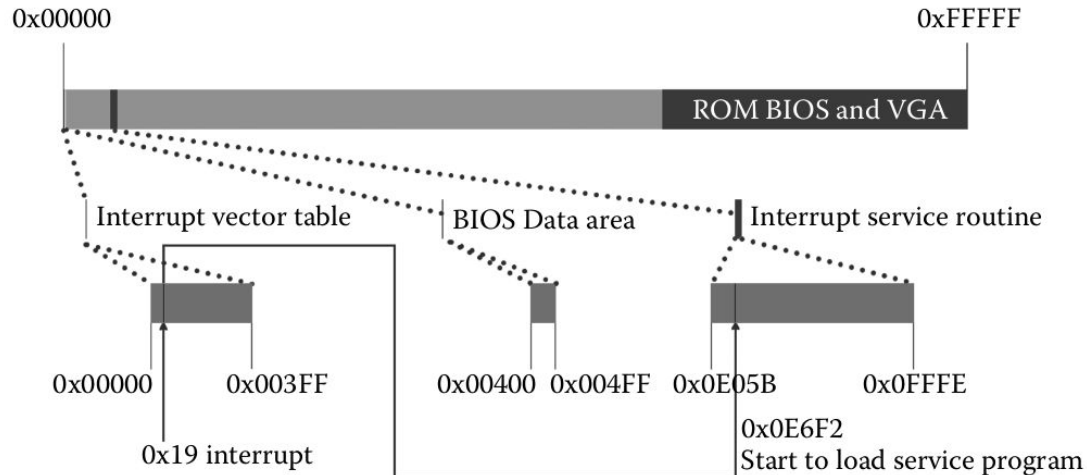
Crear la **Interrupt Vector Table** y cargar las rutinas de manejo de interrupciones en **Modo Real**: el BIOS pone la tabla de interrupciones en el inicio de la memoria 1 KB (0x00000-0x003FF), son 256 entradas de 4 bytes.

El área de datos del BIOS de unos 256 B (0x00400-0x004FF), y el servicio de atención de interrupciones (8 KB), 56 KB, después de la dirección 0x0E05B.



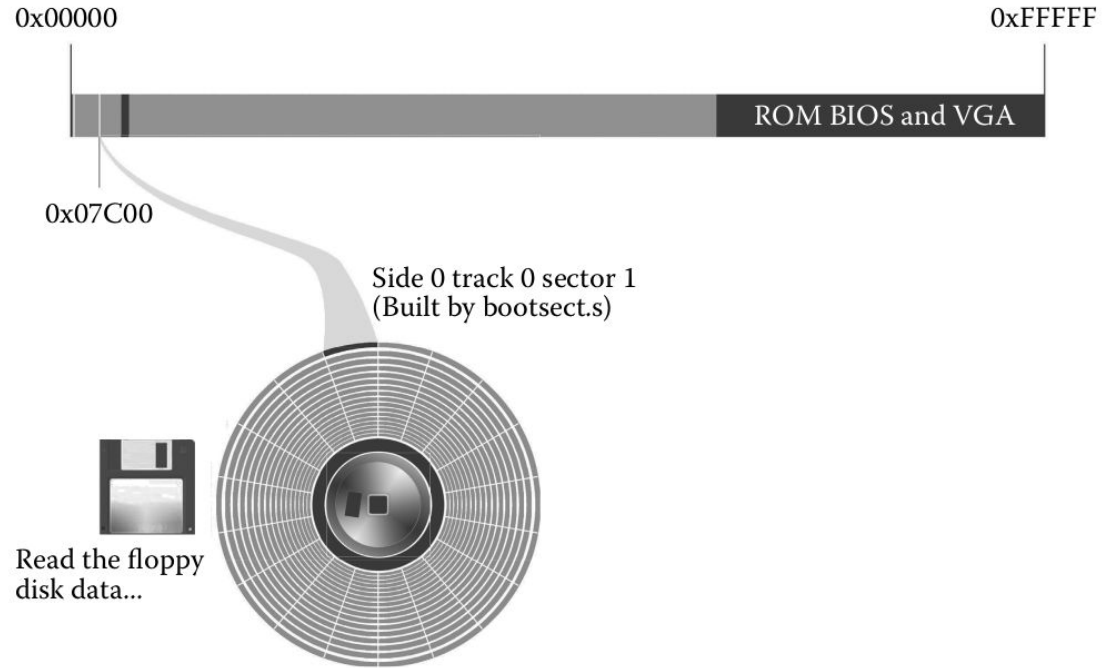
Booteo 3

La BIOS genera una interrupción 19 (INT 19) de la tabla de interrupciones la cual hace apuntar a CS:IP a 0x0E6F2.



Booteo 4

Lo cual hace ejecutar el servicio de interrupciones, el handler de dicha interrupción, que es leer el primer sector de 512 bytes del disco a memoria, y ahí termina.



Fase de Inicio del kernel



La función de arranque para el kernel (también llamado intercambiador o proceso 0) establece la gestión de memoria (tablas de paginación y paginación de memoria), detecta el tipo de CPU y cualquier funcionalidad adicional como capacidades de punto flotante, y después cambia a las funcionalidades del kernel para arquitectura no específicas de Linux, a través de una llamada a la función `start_kernel()`.

Fase de Inicio del kernel



Por lo tanto, el núcleo **inicializa los dispositivos, monta el sistema de archivos raíz** especificado por el gestor de arranque como de sólo lectura, **y se ejecuta Init (/sbin/init)**, que es designado como el primer proceso ejecutado por el sistema (PID=1). También puede ejecutar opcionalmente **initrd** para permitir instalar y cargar dispositivos relacionados (disco RAM o similar), para ser manipulados antes de que el sistema de archivos raíz está montado.

En este punto, con las interrupciones habilitadas, el programador puede tomar el control de la gestión general del sistema, para proporcionar multitarea preventiva, e iniciar el proceso para continuar con la carga del entorno de usuario en el espacio de usuario.

El Proceso de Inicio



El trabajo de **Init** es “**conseguir que todo funcione como debe ser**” una vez que el kernel está totalmente en funcionamiento. En esencia, establece y opera todo el espacio de usuario. Esto incluye:

1. la comprobación y montaje de sistemas de archivos
2. la puesta en marcha los servicios de usuario necesarios y, en última instancia, cambiar al entorno de usuario cuando el inicio del sistema se ha completado.