Sistemas Operativos
Fisop 2024

El Kernel

La abstracción User/Kernel

Conceptos clave:

- Ejecución directa vs directa limitada
- Estructura de UNIX
- System calls

Ejecución Directa

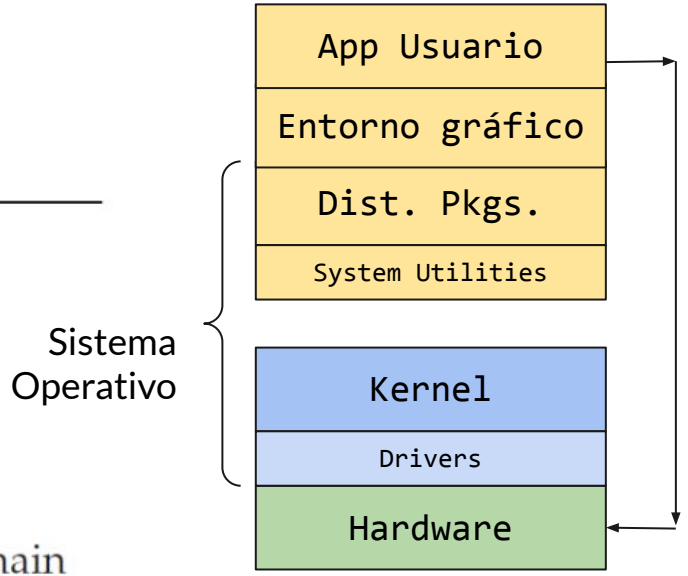
En un sistema de ejecución directa, **los programas tienen acceso completo a todo el hardware del sistema**, sin ningún tipo de control.

El sistema operativo, provee rutinas a modo de biblioteca. Actua principalmente como pegamento, pero no provee aislamiento

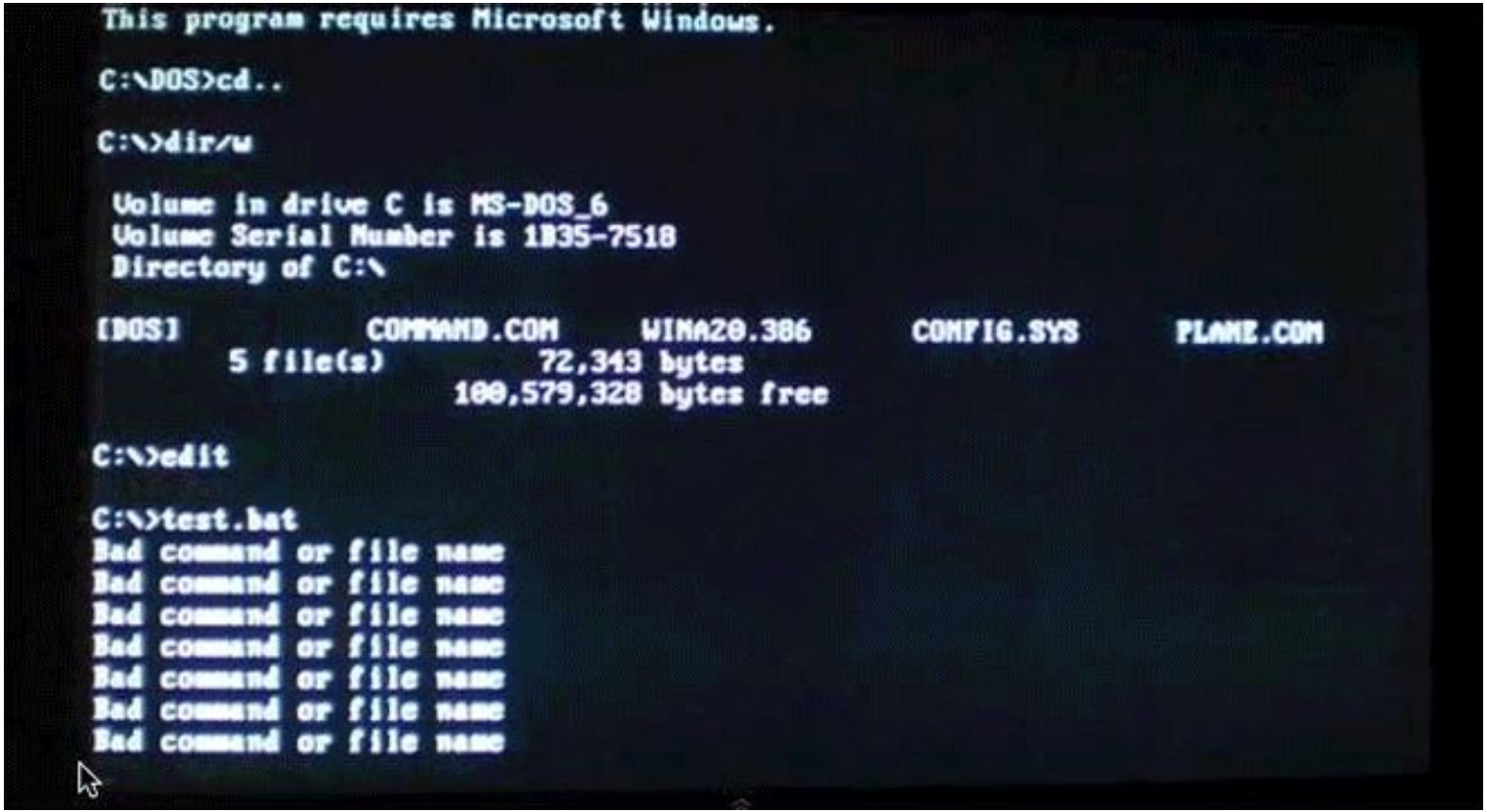
Eg. DOS

Ejecución Directa

OS	Program
Create entry for process list	
Allocate memory for program	
Load program into memory	
Set up stack with argc/argv	
Clear registers	
Execute **call** main()	
	Run main()
	Execute **return** from main
Free memory of process	
Remove from process list	

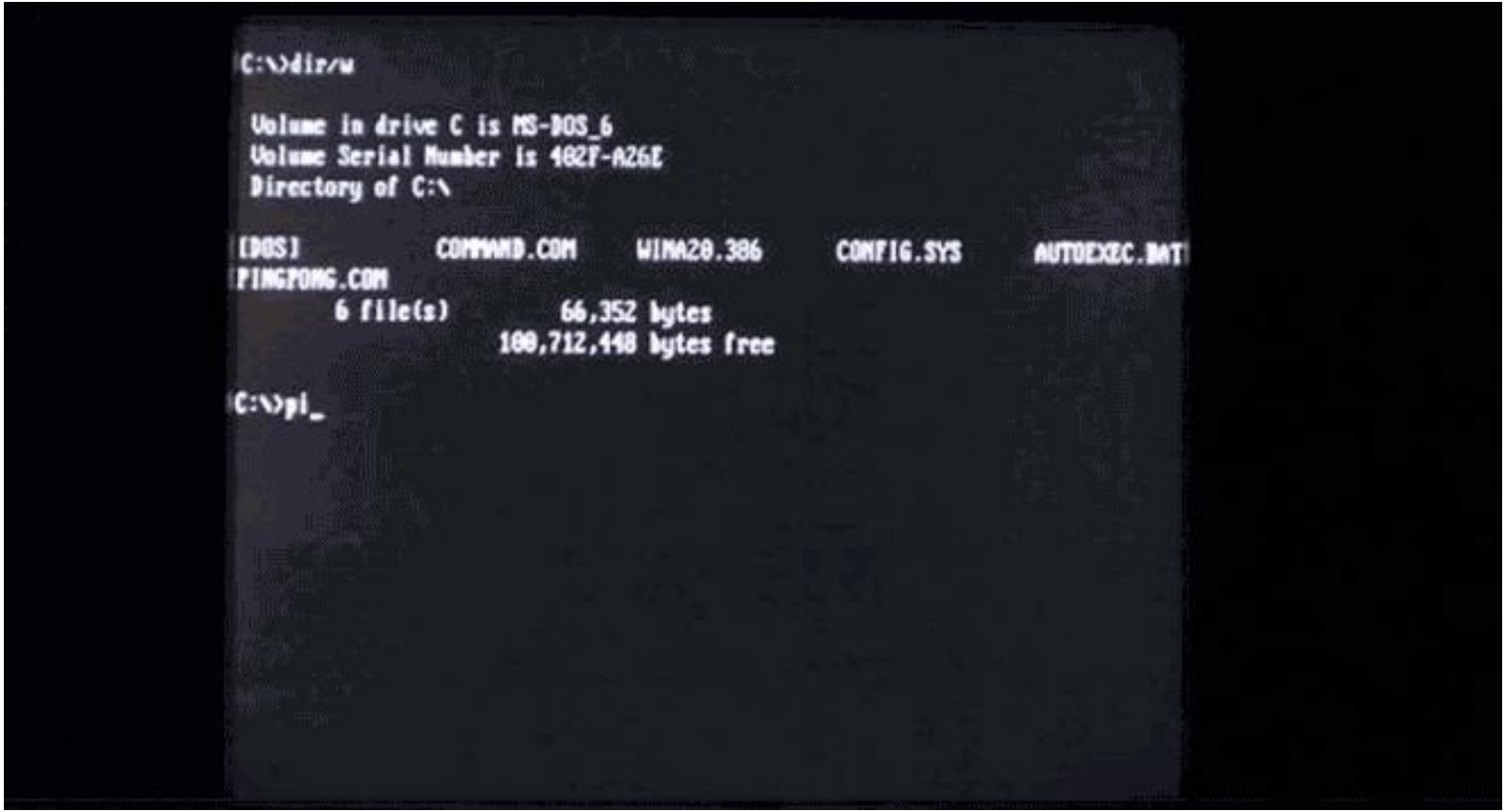


Ejecución Directa



Ejecución Directa

Ejecución Directa



Ejecución Directa

Ejecución Directa

Problema:

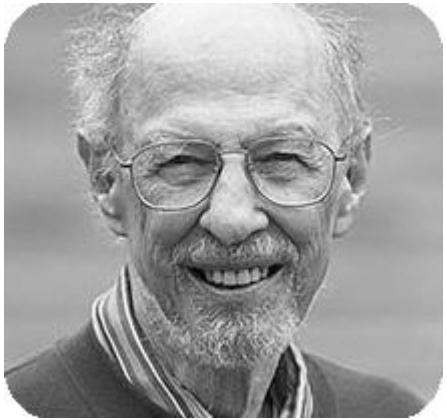
La ejecución directa no provee aislamiento y protección.

Ejemplos:

- Nada garantiza que un programa no modifique a otro programa (o al sistema operativo mismo)
- Acceso directo al storage permanente: nada garantiza que un programa no dañe el filesystem.
- Nada garantiza que un programa termine, se interrumpa, etc.

Ejecución Directa Limitada

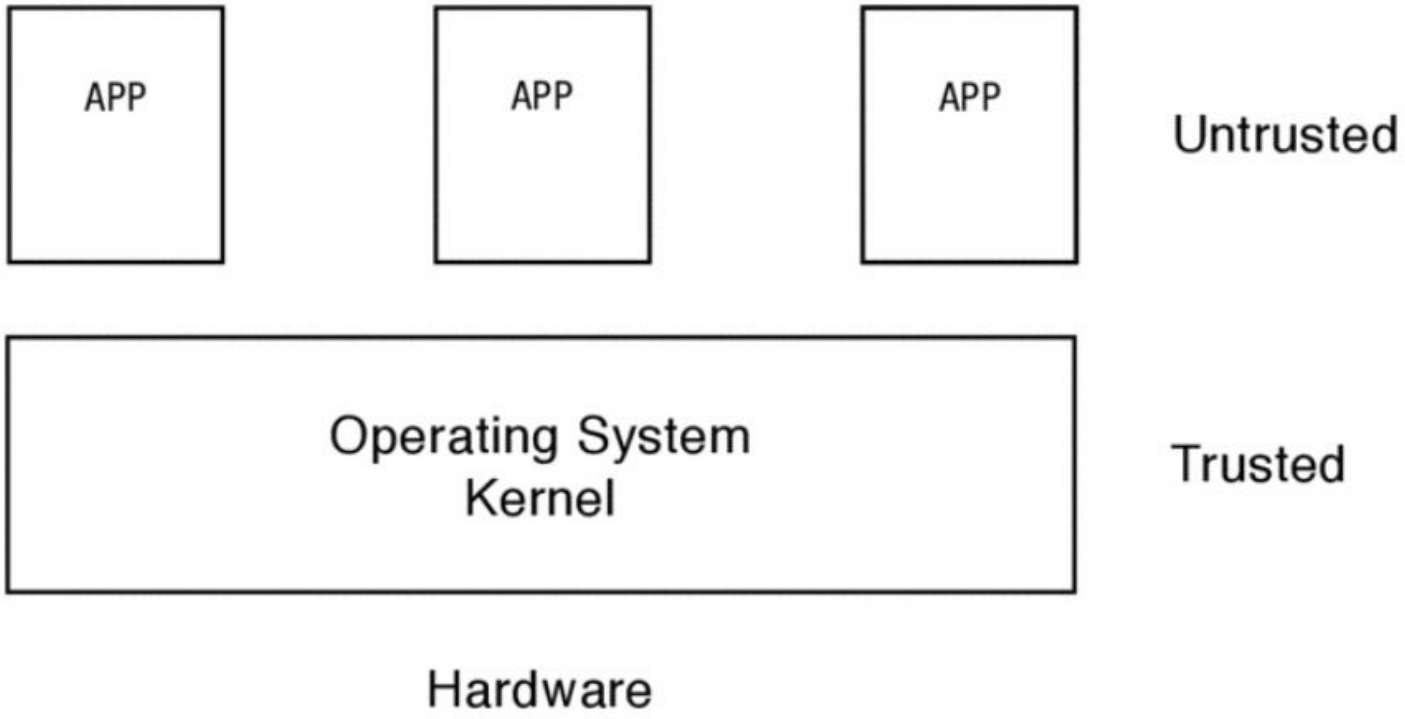
1. El hardware va a limitar ciertas operaciones.
2. Un programa especial y privilegiado va a arbitrar las operaciones riesgosas



Fernando J. Corbató
Pionero del proyecto Multics wiki

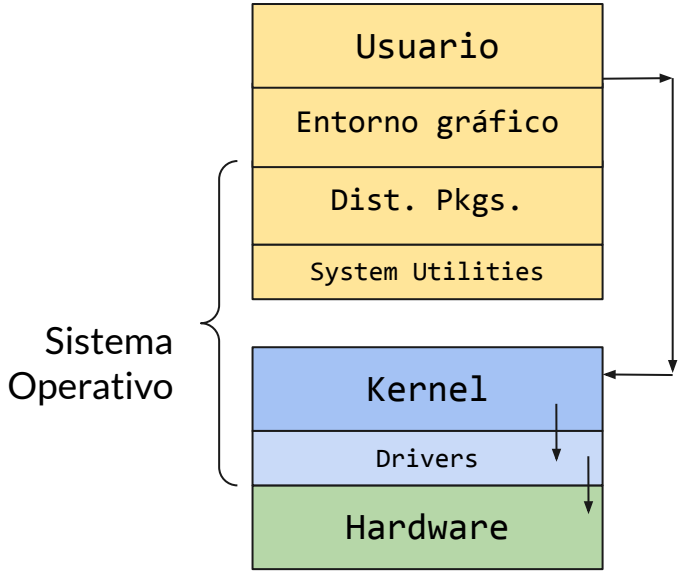
El rol central de un sistema operativo es la protección.

Kernel Land y User Land

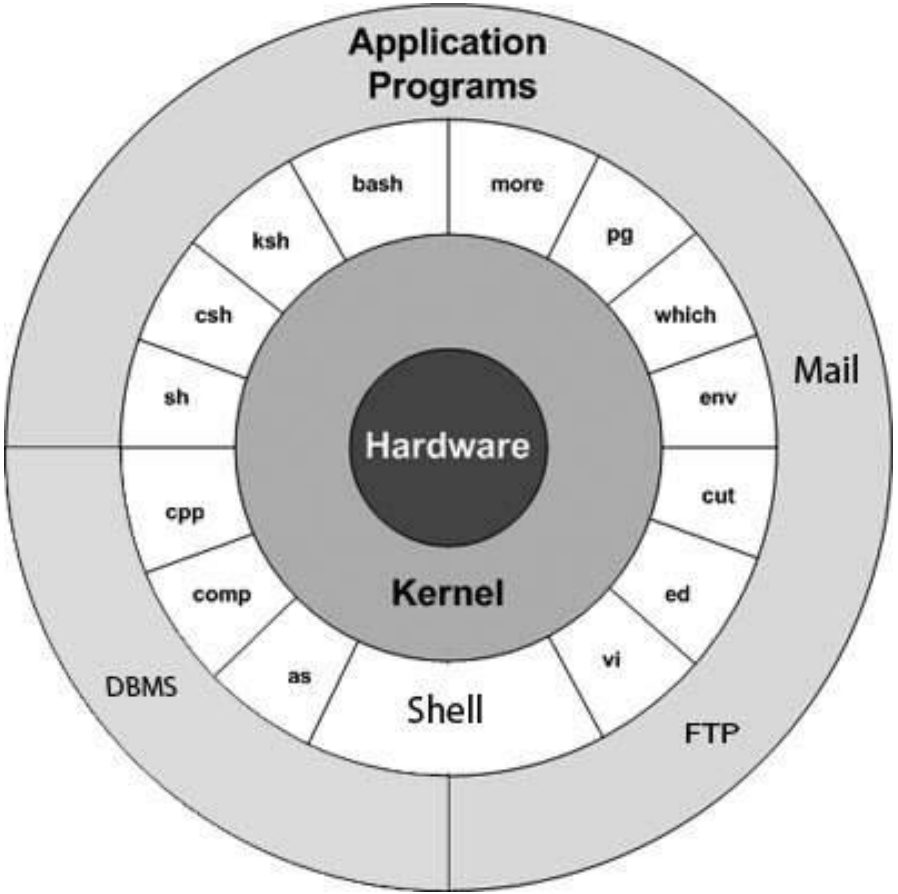


OS @ boot (kernel mode)	Hardware	Program (user mode)
initialize trap table		
	remember address of... syscall handler	
OS @ run (kernel mode)	**Hardware**	**Program (user mode)**
Create entry for process list		
Allocate memory for program		
Load program into memory		
Setup user stack with argv		
Fill kernel stack with reg/PC		
return-from-trap		
	restore regs (from kernel stack)	
	move to user mode	
	jump to main	
		Run main()
		...
		Call system call **trap** into OS
	save regs (to kernel stack)	
	move to kernel mode	
	jump to trap handler	
Handle trap		
Do work of syscall		
return-from-trap		
	restore regs (from kernel stack)	
	move to user mode	
	jump to PC after trap	
		...
		return from main
		trap (via `exit()`)
Free memory of process		
Remove from process list		

Limitar la Ejecución Directa



Estructura del sistema operativo UNIX



User-space vs Kernel space

User space

- Una **aplicación** puede ejecutarse **solo en modo usuario.**
- No puede ejecutar instrucciones privilegiadas.
- Se dice que se ejecuta en **user space**.

Kernel space

- **Solo el kernel** se ejecuta en modo supervisor.
- Puede ejecutar instrucciones privilegiadas.
- Se dice que se ejecuta en **kernel space**.

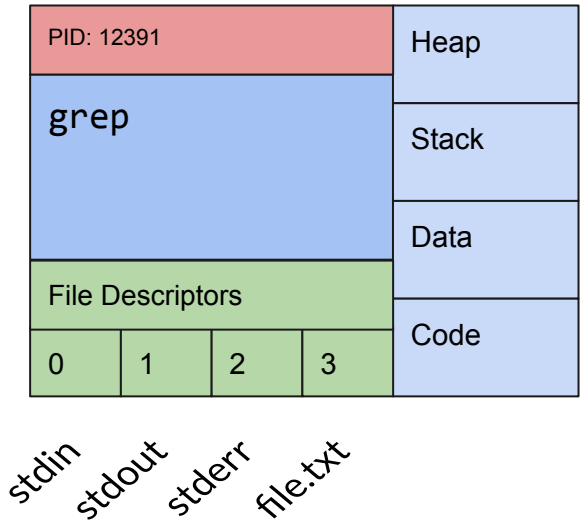
User Space

User-land: espacio donde viven las aplicaciones de usuario, a estas se las denominan **procesos**.

Cada proceso o programa en ejecución posee memoria con las instrucciones,los datos, el stack y el heap.

User Space

El habitante de User Land: **el proceso**



¿Y qué pasa en User space?

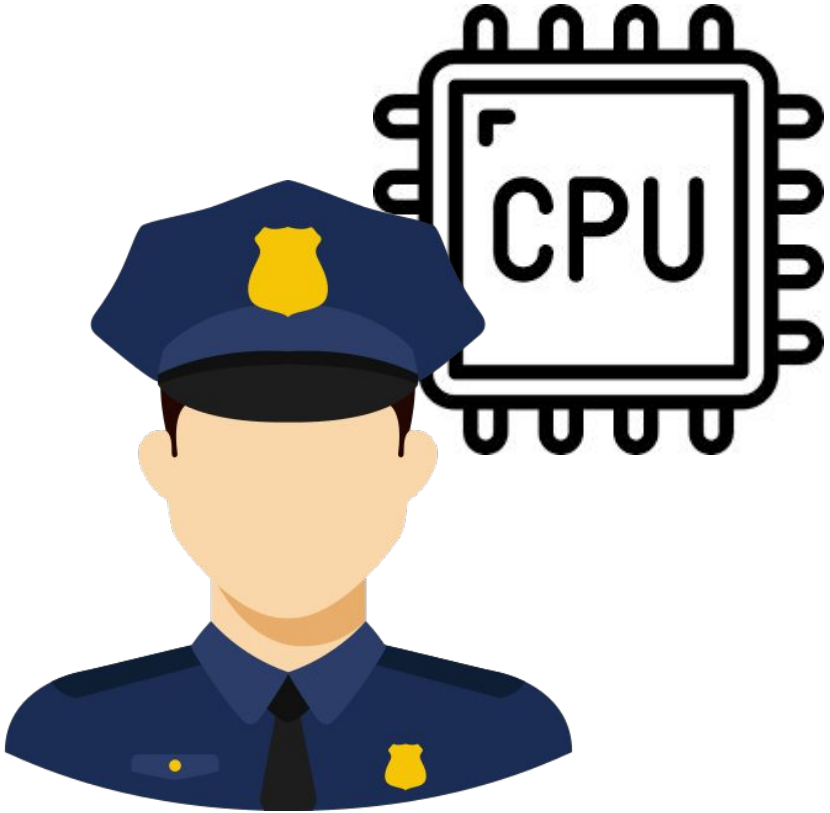
Es donde se ejecutan todos los programas (procesos)!

Los procesos se ejecutan en un **contexto:**

- **Aislado**
- **Protegido**
- **Restringido**

Mediante el uso de funciones que se encuentran en bibliotecas pueden utilizar los servicios de acceso al hardware o **recursos que el kernel proporciona.**

User-space vs Kernel space

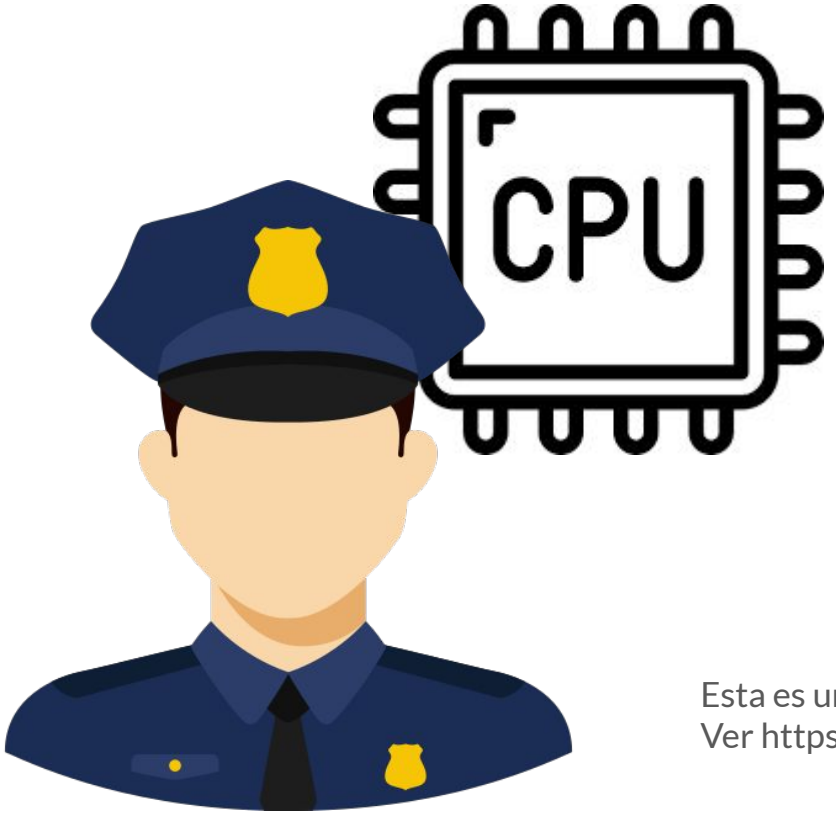


Policía de execution modes

¿Quien, en ultima instancia, controla que los programas (user-mode) se comporten correctamente?

El hardware

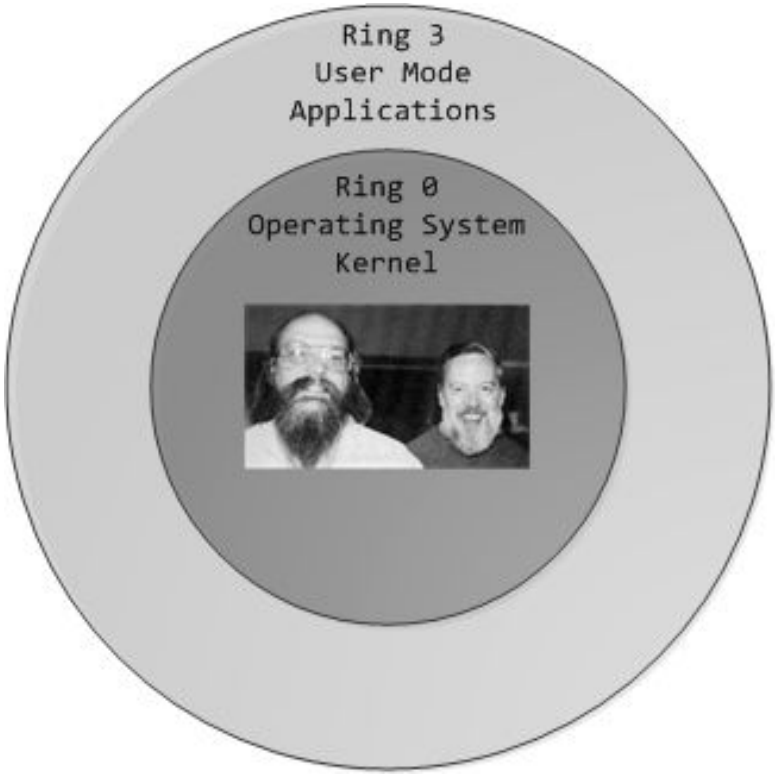
User-space vs Kernel space



- El hardware provee el **mecanismo de protección**
- El software (kernel) determina la **política de protección** y configura el hardware.

Esta es una instancia del principio de separación mecanismo/politica.
Ver https://en.wikipedia.org/wiki/Separation_of_mechanism_and_policy

User-space vs Kernel space



Las CPUs proporcionan soporte de hardware para un aislamiento fuerte. Por ejemplo:

RISC-V tiene tres modos en los que la CPU puede ejecutar instrucciones: **modo máquina, modo supervisor y modo usuario**.

X86 tiene cuatro modos en los que la CPU puede ejecutar instrucciones, solo usa 2: **modo supervisor y modo usuario**.

En modo **supervisor**, la CPU puede ejecutar instrucciones privilegiadas: por ejemplo, habilitar y deshabilitar interrupciones, leer y escribir en el registro que contiene la dirección de una tabla de páginas, etc.

User-space vs Kernel space

Una aplicación que quiere invocar una función del Kernel (por ejemplo, la syscall read en xv6) debe realizar una transición al Kernel; una aplicación no puede invocar directamente una función del Kernel.

Las CPUs proporcionan una instrucción especial que cambia la CPU del modo usuario al modo supervisor y entra en el Kernel en un punto de entrada especificado por el Kernel. (RISC-V proporciona la instrucción ecall para este propósito, x86 int 0x80.)

Una vez que la CPU cambio al modo supervisor, el Kernel puede entonces validar los argumentos de la llamada al sistema (por ejemplo, verificar si la dirección pasada a la llamada al sistema es parte de la memoria de la aplicación), decidir si la aplicación tiene permisos para realizar la operación solicitada (por ejemplo, verificar si la aplicación tiene permisos para escribir en el archivo especificado) y luego denegarla o ejecutarla.

Es importante que el Kernel controle el punto de entrada para las transiciones al modo supervisor; Si la aplicación pudiera decidir el punto de entrada al núcleo, una aplicación maliciosa podría, por ejemplo, ingresar al núcleo en un punto donde se omite la validación de argumentos.

Modo Dual de Operaciones

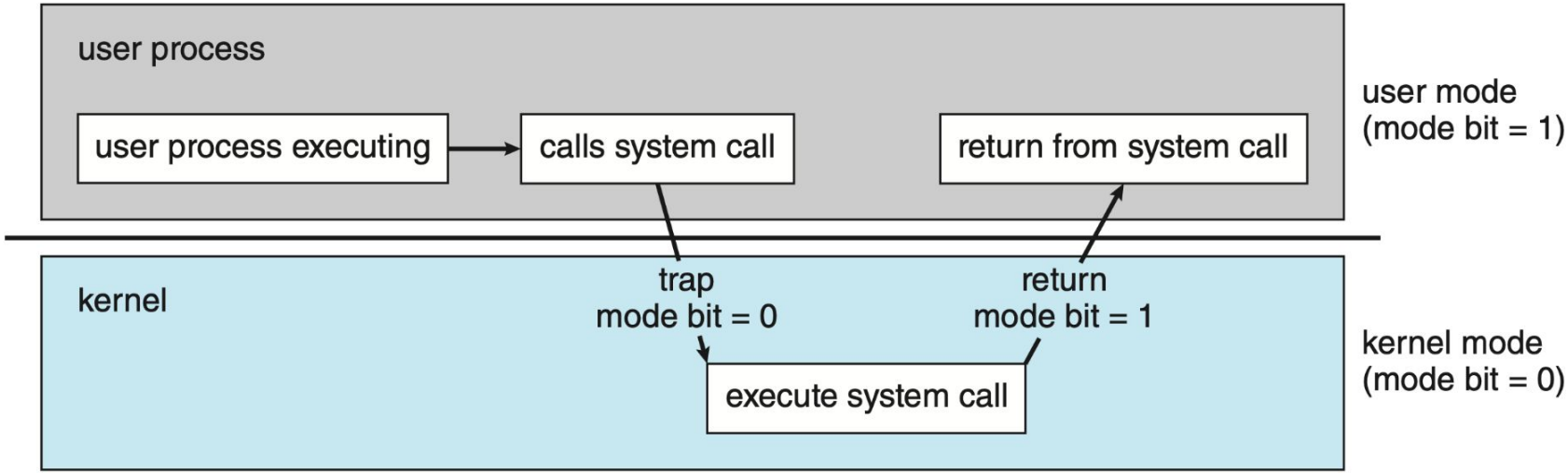
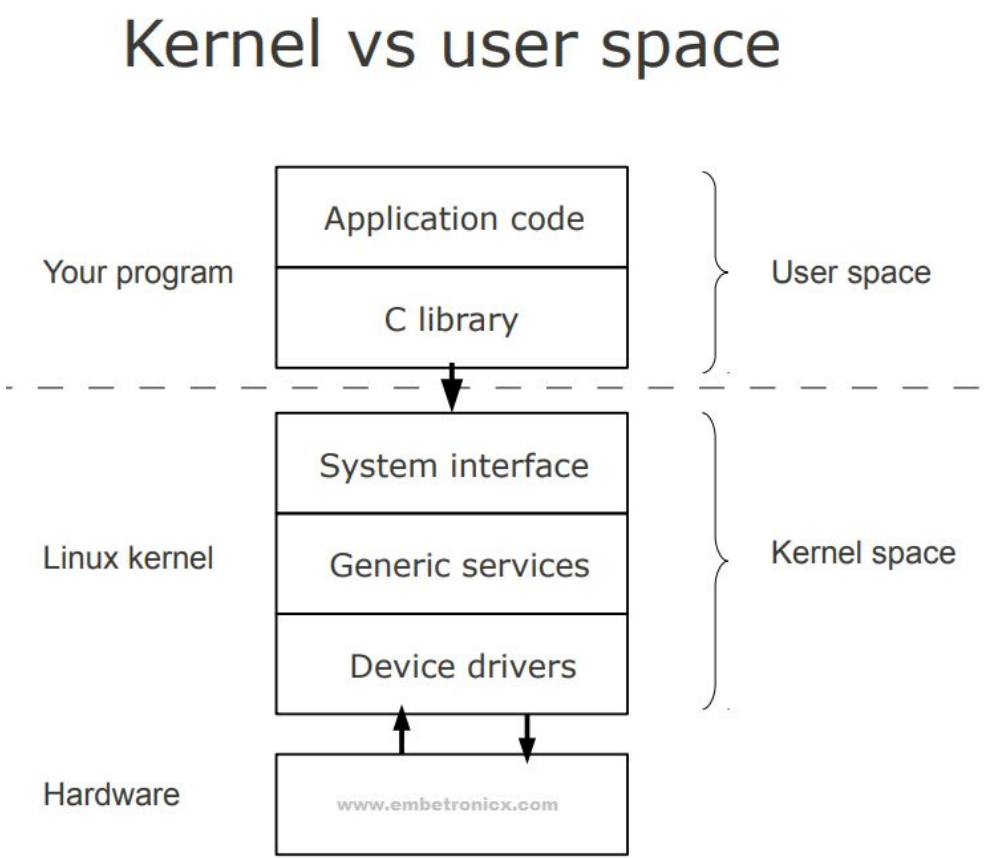


Figure 1.13 Transition from user to kernel mode.

User-space vs Kernel space



System Calls

Una **system call (llamada al sistema) es un punto de entrada controlado al kernel**, permitiendo a un proceso solicitar que el kernel realice alguna operación en su nombre [KER](cap. 3).

El kernel expone una gran cantidad de servicios accesibles por un programa vía el **Application Programming Interface (API) de system calls**.

Una Syscall es un servicio que provee el kernel

System Calls

Algunas características generales de las system calls son:

Una *system call* **cambia el modo del procesador de user mode a kernel mode**, por ende la CPU podrá acceder al área protegida del kernel.

El **conjunto de system calls es fijo**. Cada system call está identificada por un único número, que por supuesto no es visible al programa, éste sólo conoce su nombre.

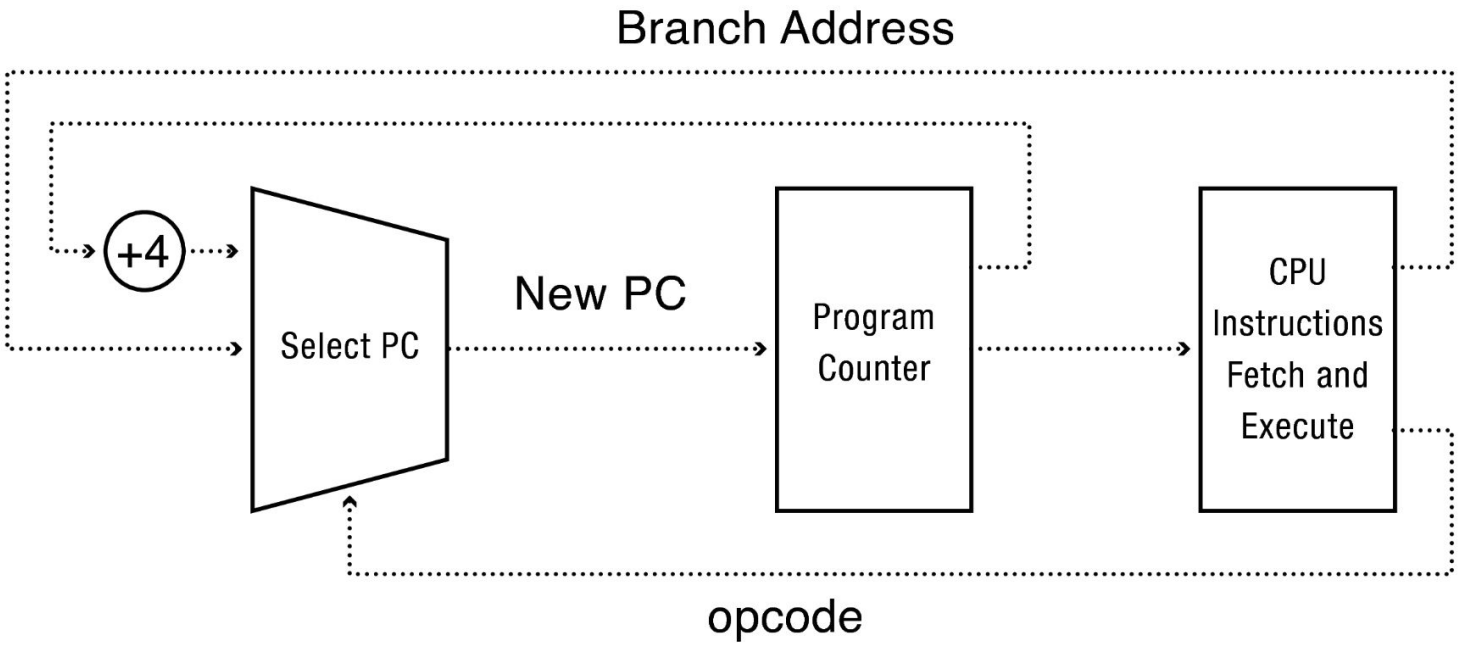
Cada system call debe tener un conjunto de parámetros que especifican información que debe ser transferida desde el user space al kernel space.

Mecanismos de protección del hardware

Conceptos clave:

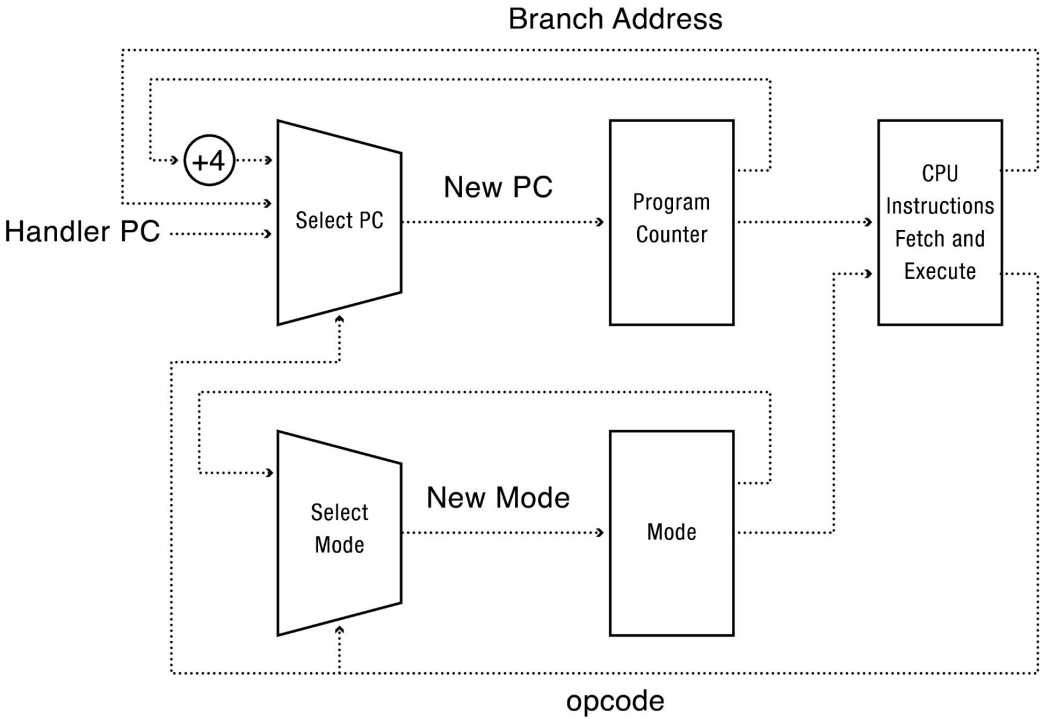
- Instrucciones privilegiadas
- Protección de memoria
- Timer interrupts

Ejecución Directa



Modo Dual de Operaciones

Existen ciertos estados en el que el procesador puede encontrarse.



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

- Instrucciones Privilegiadas
- Protección de Memoria
- Timer Interrupts

Modos en Risc-V

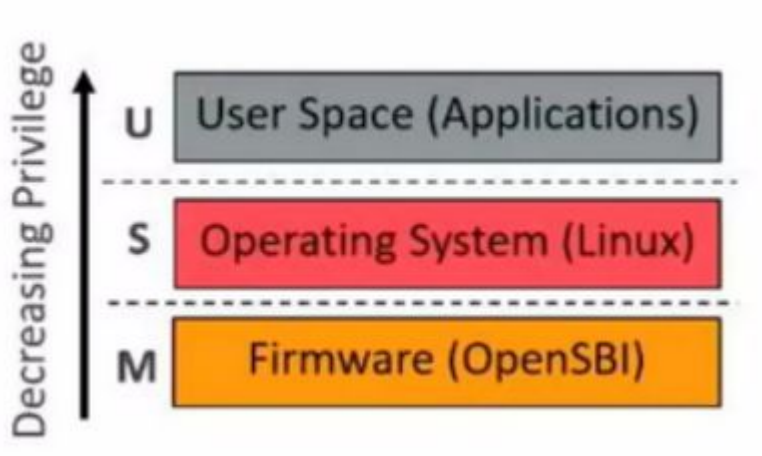
Risc-V:

Cada nivel de privilegio puede ejecutar distintas instrucciones:

Machine:wfi

Supervisor:crrr, crrw,csrrw

User: la, li, etc.

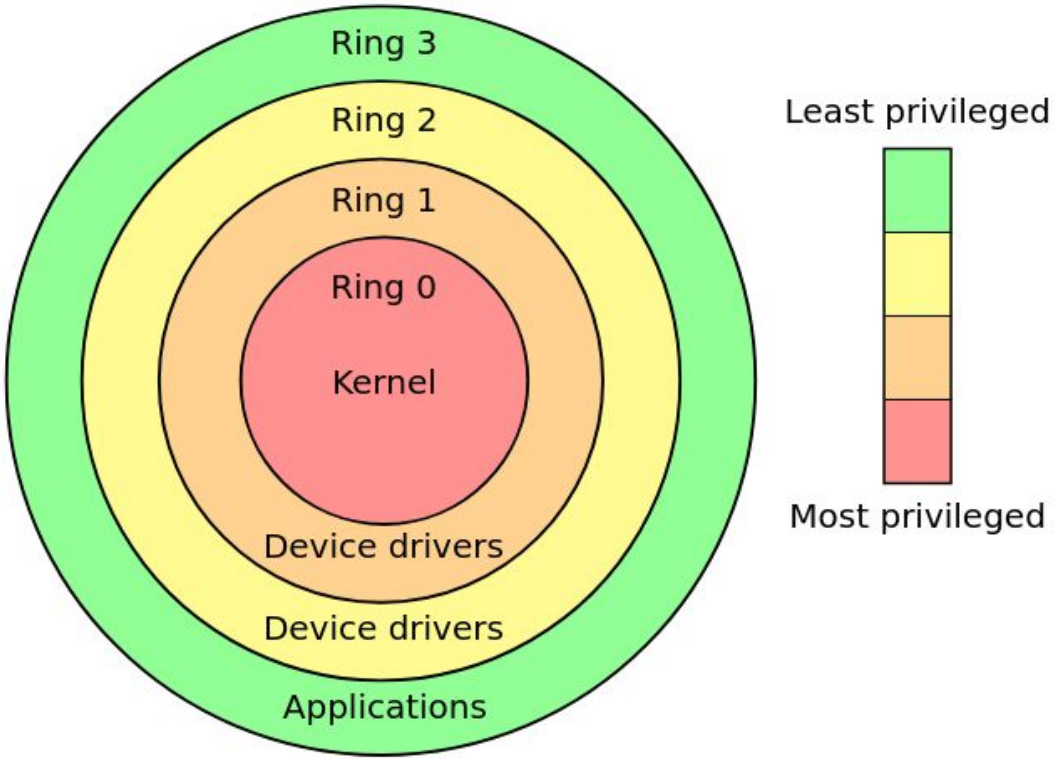


Level	Encoding	Name	Abbreviation
0	00	User/Application	U
1	01	Supervisor	S
2	10	*Reserved*	
3	11	Machine	M

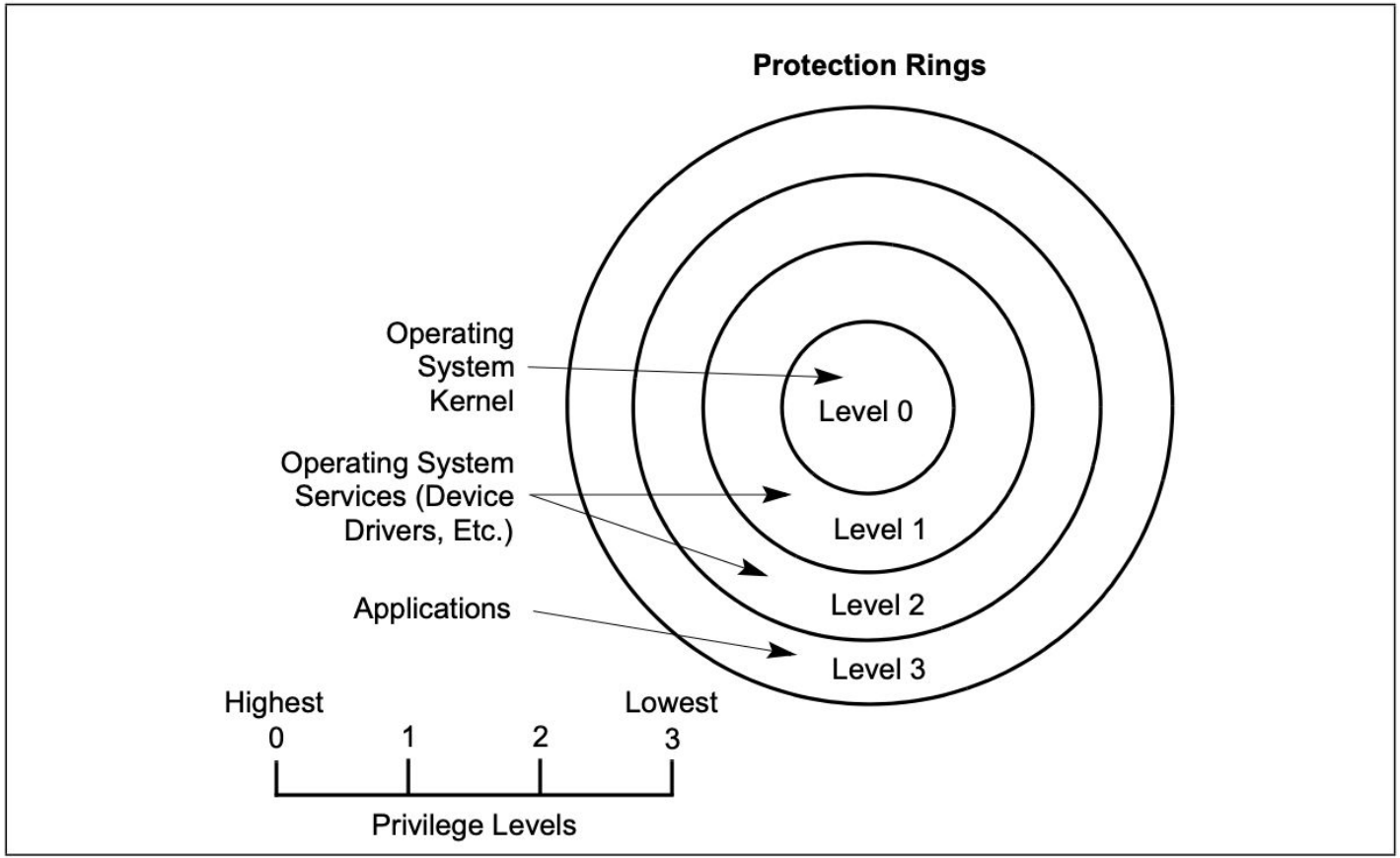
Modos en x86

X86

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



Modos en x86


Protection Rings
Operating System Kernel
Operating System Services (Device Drivers, Etc.)
Applications
Level 0
Level 1
Level 2
Level 3
Highest
Lowest
0
1
2
3
Privilege Levels

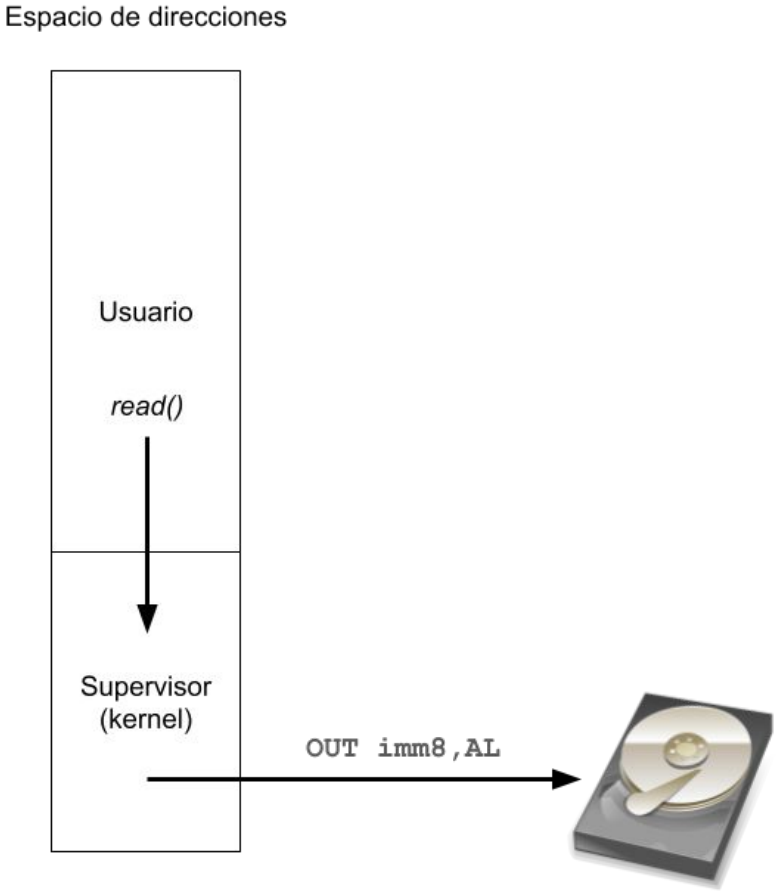
Instrucciones Privilegiadas

Definición informal

Toda instrucción que el modo usuario no puede ejecutar.

(Y que si insiste en ejecutar, el procesador genera una excepción, o falla de alguna manera)

Ejemplo. Leer el disco en x86



Ejemplo. Leer el disco en x86

Ejemplo:

Instruccion OUT

```
OUT —— Output to Port : Transfiere un byte de datos o
una palabra de datos desde el registro (AL, AX o EAX),
dado como el segundo operando, al puerto de salida
direccionado por el primer operando.

Eg.

   OUT imm8,AL
```

IOPL

Controla el modo mínimo necesario para escribir puertos

I/O Privilege level

Ocupa los bits 12 y 13 en el registro FLAGS.

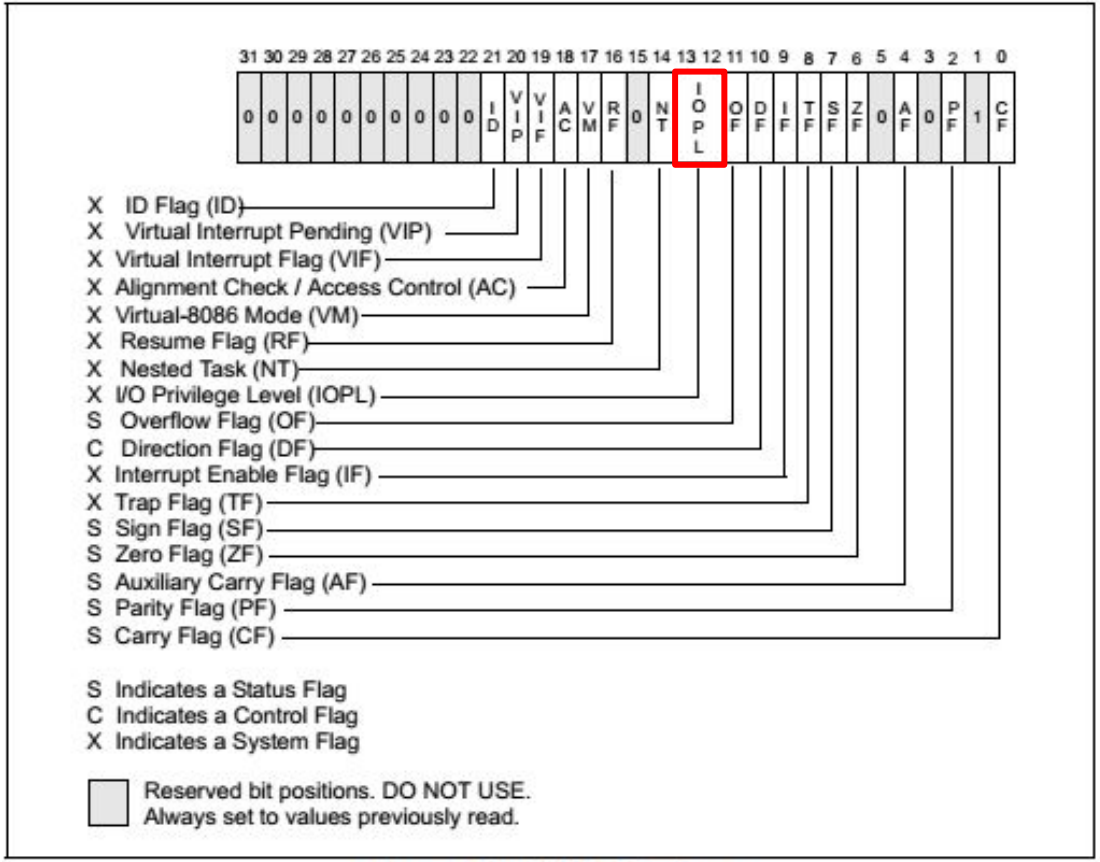


Figure 3-8. EFLAGS Register

Modo protegido en x86 (simplificado)

Pseudocódigo de la instrucción OUT

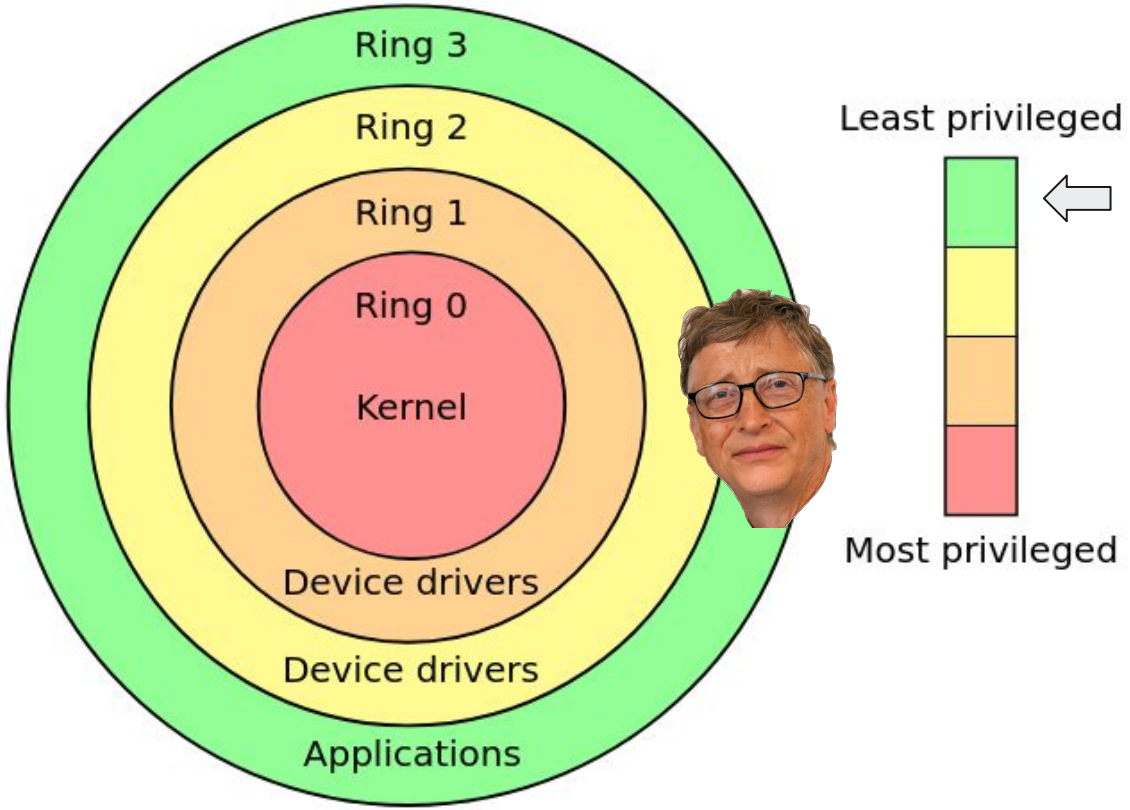
```
IF (PE = 1) AND ((VM = 1) OR (CPL > IOPL))
THEN (* Virtual 8086 mode, or protected mode with CPL > IOPL *)
 IF NOT I-O-Permission (DEST, width(DEST))
 THEN #GP(0);
 FI;
FI;
[DEST] ← SRC; (* I/O address space used *)
```

En ROJO la validacion de permisos, en AZUL la operación en sí misma

Modo protegido en x86 (más simplificado)

Asumiendo que IOPL suele ser 0, porque solo el modo kernel puede ejecutarlo

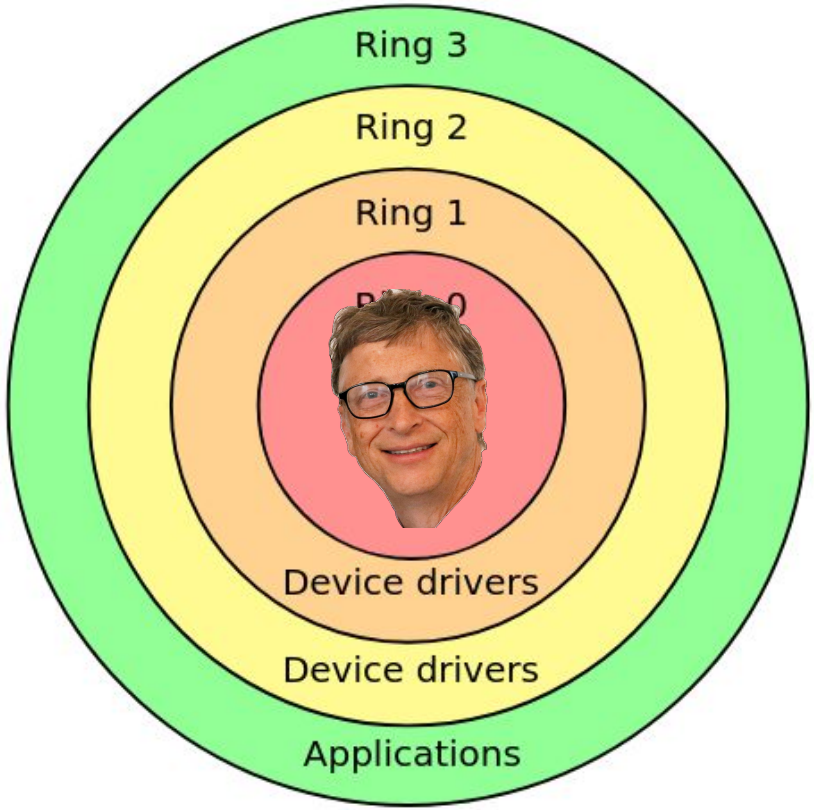
```
IF CPL > 0
THEN
    EXCEPTION!!!
ELSE
    [DEST] ← SRC;
FI
```

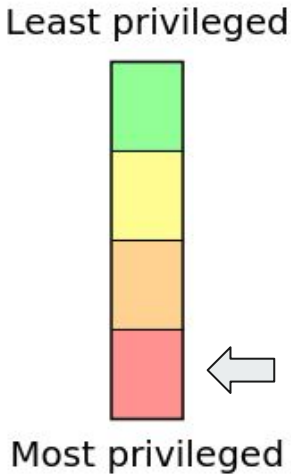


Modo protegido en x86 (más simplificado)

Asumiendo que IOPL suele ser 0, porque solo el modo kernel puede ejecutarlo

```
IF CPL > 0
THEN
    EXCEPTION!!!
ELSE
    [DEST] ← SRC;
FI
```



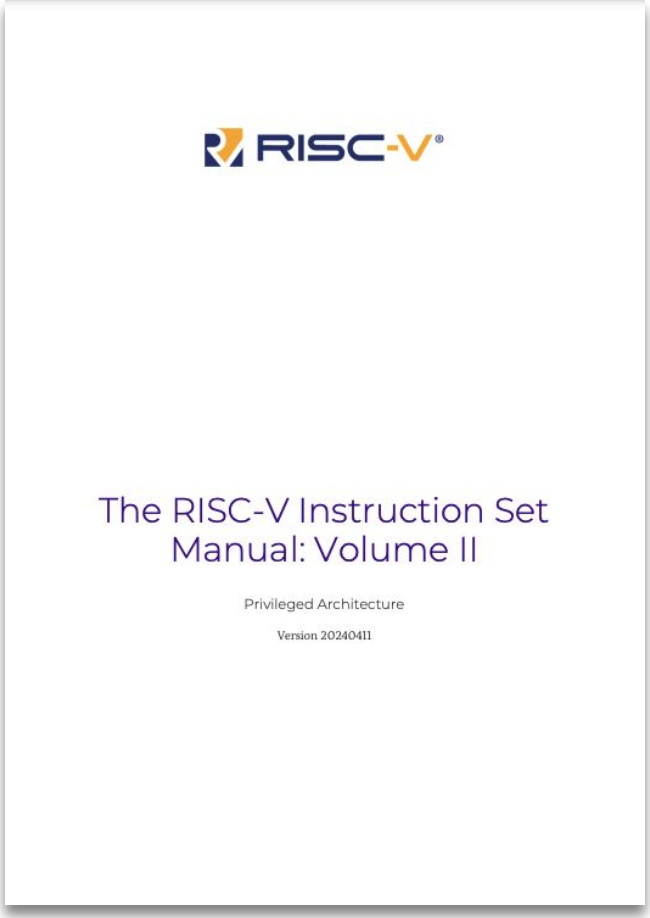


Modo protegido en x86 (simplificado)

Para pasar de Modo Usuario (CPL 3) a Modo Supervisor (CPL 0) mediante una System Call hay un protocolo bien definido que los programas ejecutan

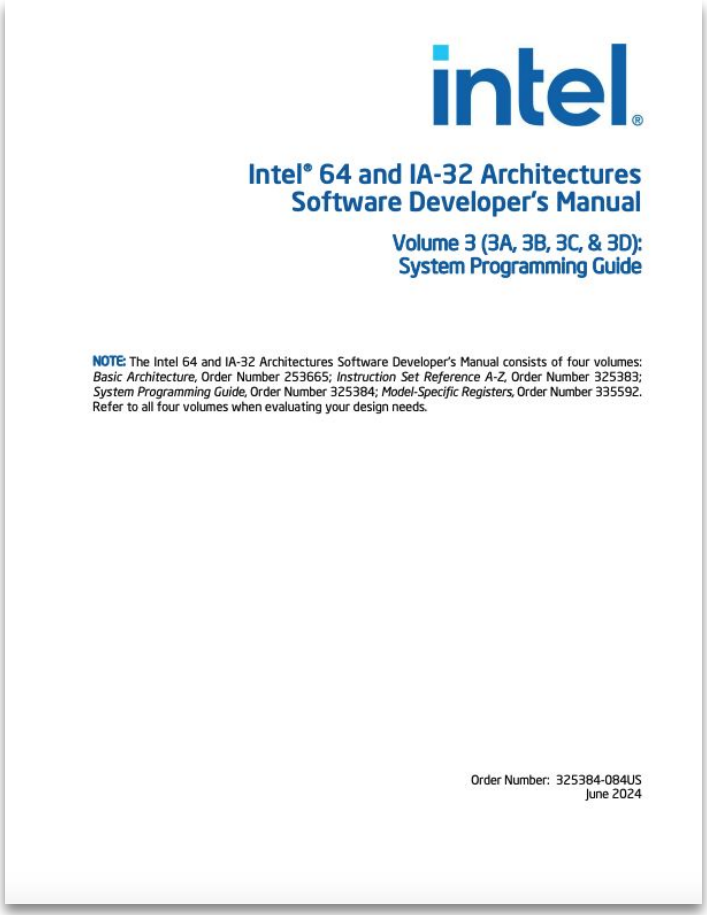
Instrucciones Privilegiadas

Un manual entero de instrucciones privilegiadas para RISC-V



Instrucciones Privilegiadas

Un manual entero (y gigante) de instrucciones privilegiadas para x86



intel®

Intel® 64 and IA-32 Architectures
Software Developer's Manual

Volume 3 (3A, 3B, 3C, & 3D):
System Programming Guide

NOTE: The Intel 64 and IA-32 Architectures Software Developer's Manual consists of four volumes: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-Z*, Order Number 325383; *System Programming Guide*, Order Number 325384; *Model-Specific Registers*, Order Number 335592. Refer to all four volumes when evaluating your design needs.

Order Number: 325384-084US
June 2024

x86: Ejemplo 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.

Ejemplos: Instrucciones Privilegiadas

- **MOV a/desde CR0-CR4 (Control Registers):** Movimientos de datos hacia o desde los registros de control CR0, CR2, CR3 y CR4. Estos registros controlan aspectos críticos del procesador como el modo de protección, paginación, y depuración.

- **HLT (Halt):** Detiene la ejecución del procesador hasta que ocurra una interrupción. Esta instrucción es usada típicamente en sistemas operativos para poner al procesador en un estado de espera.

- **LGDT (Load Global Descriptor Table):** Carga la dirección de la Global Descriptor Table (GDT) en el registro GDTR. Este registro es crucial para la administración de la memoria y la protección de segmentos en sistemas operativos que utilizan la memoria segmentada.

Ejemplos: Instrucciones No Privilegiadas

- **MOV (Move Data):** Esta instrucción se utiliza para mover datos de una ubicación a otra. Es una de las instrucciones más básicas y se puede utilizar para mover datos entre registros, entre registros y memoria, o entre puertos de E/S.
- **ADD, SUB (Arithmetic Operations):** Estas instrucciones realizan operaciones aritméticas básicas como suma y resta en registros o entre registros y memoria.
- **AND, OR, XOR, NOT (Logical Operations):** Estas instrucciones realizan operaciones lógicas bit a bit sobre registros o entre registros y memoria.
- **JMP, JE, JNE, JG, JL (Jump Instructions):** Instrucciones de salto que permiten alterar el flujo de ejecución del programa basado en condiciones. Estas instrucciones no son privilegiadas porque solo afectan el flujo de control dentro del programa del usuario.
- **PUSH, POP (Stack Operations):** Manipulan la pila al guardar (`PUSH`) o restaurar (`POP`) valores desde ella. Estas instrucciones son esenciales para la gestión del contexto y el flujo de control en los programas.
- **NOP (No Operation):** Instrucción que no realiza ninguna operación, solo avanza el puntero de instrucción. Puede ser usada para alineación de código o para crear retrasos intencionales en la ejecución.
- **INT n (Software Interrupt):** Genera una interrupción de software. Aunque la propia instrucción no es privilegiada, su manejo depende del vector de interrupción, que puede implicar código privilegiado.

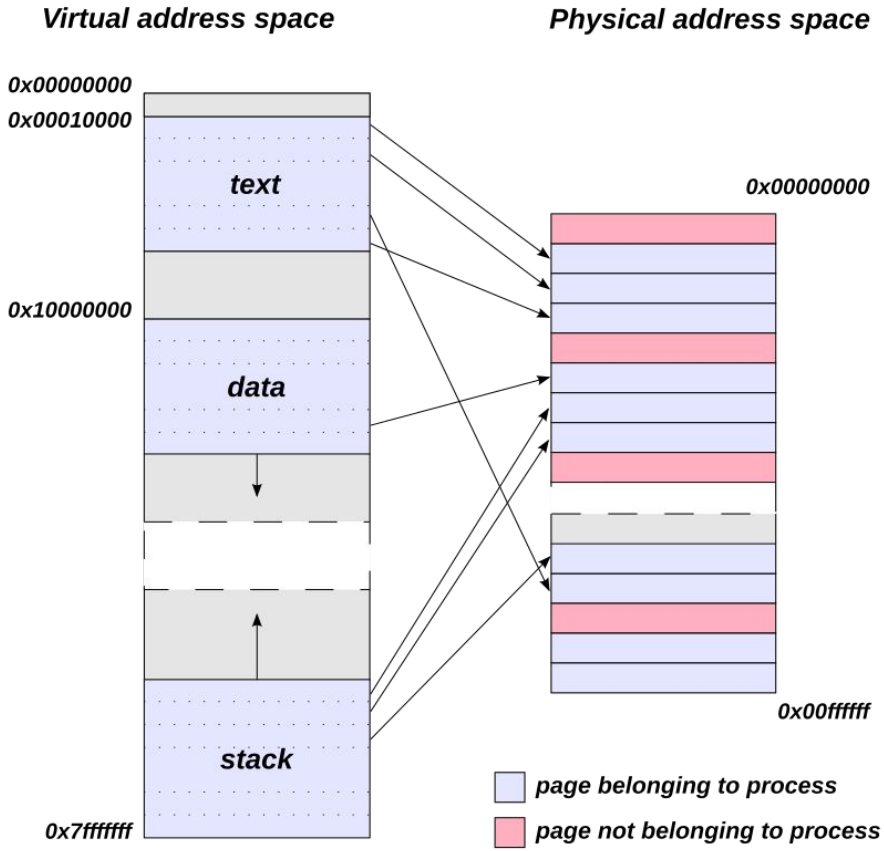
Ejemplos: Instrucciones No Privilegiadas

Mención especial: INT no es privilegiada!

- **INT n (Software Interrupt):** Genera una interrupción de software. Aunque la propia instrucción no es privilegiada, su manejo depende del vector de interrupción, que puede implicar código privilegiado.

Se va a usar para implementar la transición de las system calls

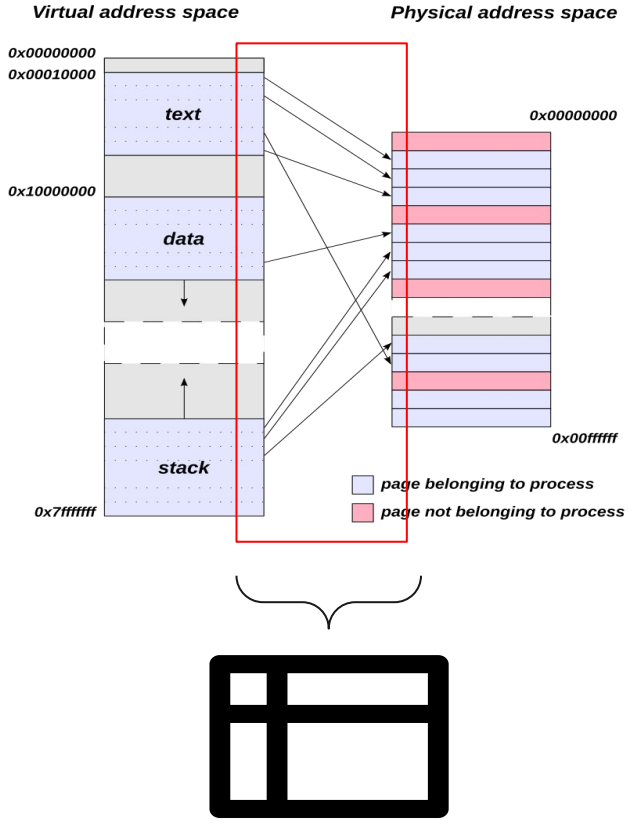
Protección de Memoria



Protección de Memoria

El mapeo se realiza mediante una tabla de páginas.
La tabla contiene para cada página mapeada:

- La correspondencia Virtual -> Física
- Flags y configuración de cada página



Page Tables

Protección de Memoria

Un registro de la tabla de páginas (RISC-V)

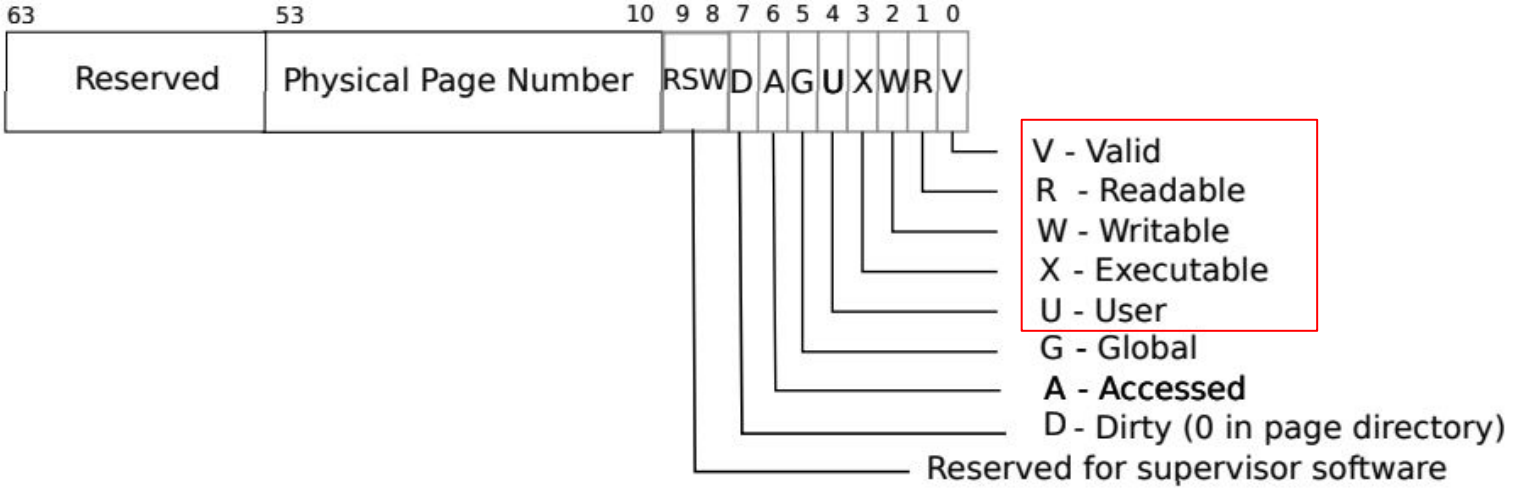


Figure 3.2: RISC-V address translation details.

Protección de Memoria

- Acceder a una página no de usuario → Hay excepción
- Escribir a una página de solo lectura → Hay excepción
- Acceder a una página no mapeada → Hay excepción
- Ejecutar una página no ejecutable → Hay excepción

Protección de Memoria

En general:

Hacer cualquier cosa como usuario que no está permitida con la memoria →

Hay excepción!!!

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 processo de usuario al Kernel

Timer Interrupts (Interrupciones por temporizador)

Veamoslo !

sudo cat /proc/timer_list

Modos de Transferencia

Conceptos clave:

- Interrupciones y Excepciones
- System Calls

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

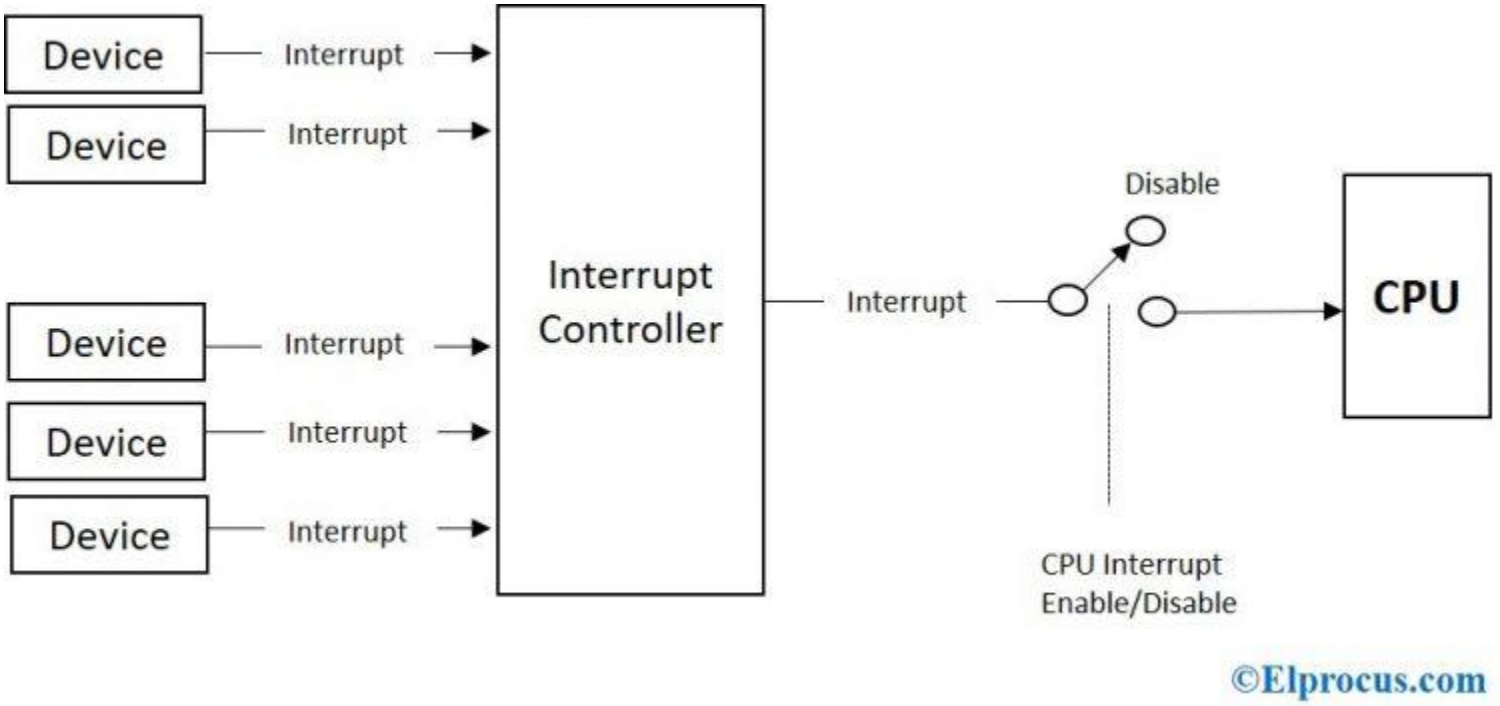
Cosas que inducen un cambio de User Mode a Kernel Mode

- Interrupciones (dispositivos I/O y el timer)
- Excepciones del Procesador
- System Calls

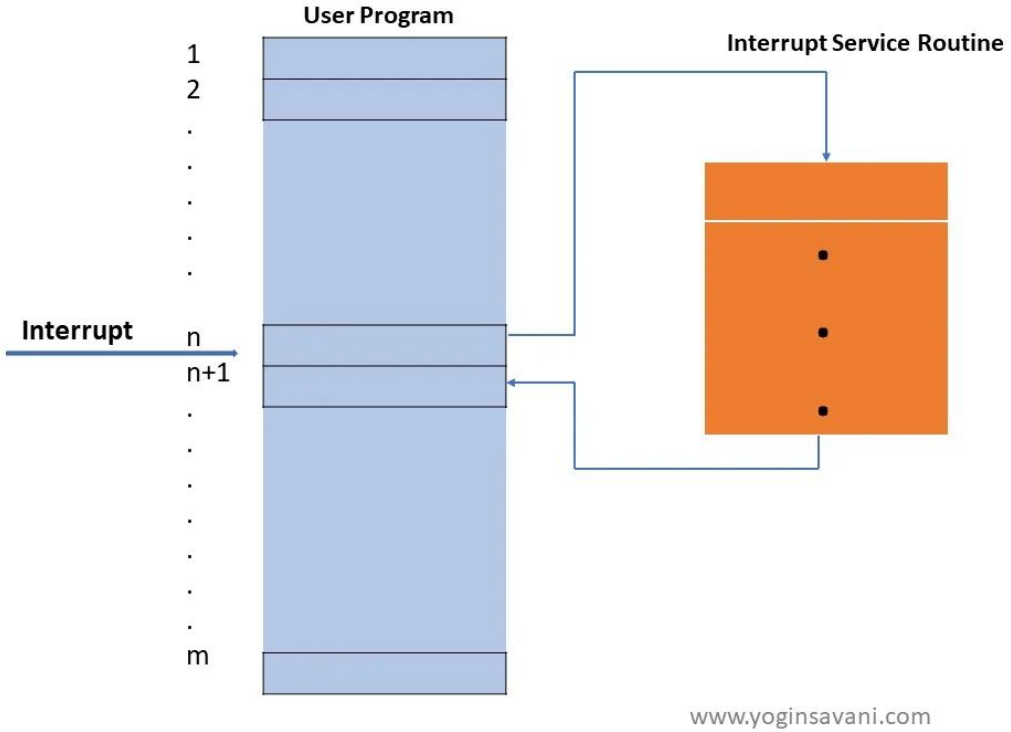
Interrupciones

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

Interrupciones



Interrupciones



La transición a la interrupción en general, causa un cambio de modo User -> Kernel

Interrupciones

Cada interrupción tiene un número asociado

```
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
```

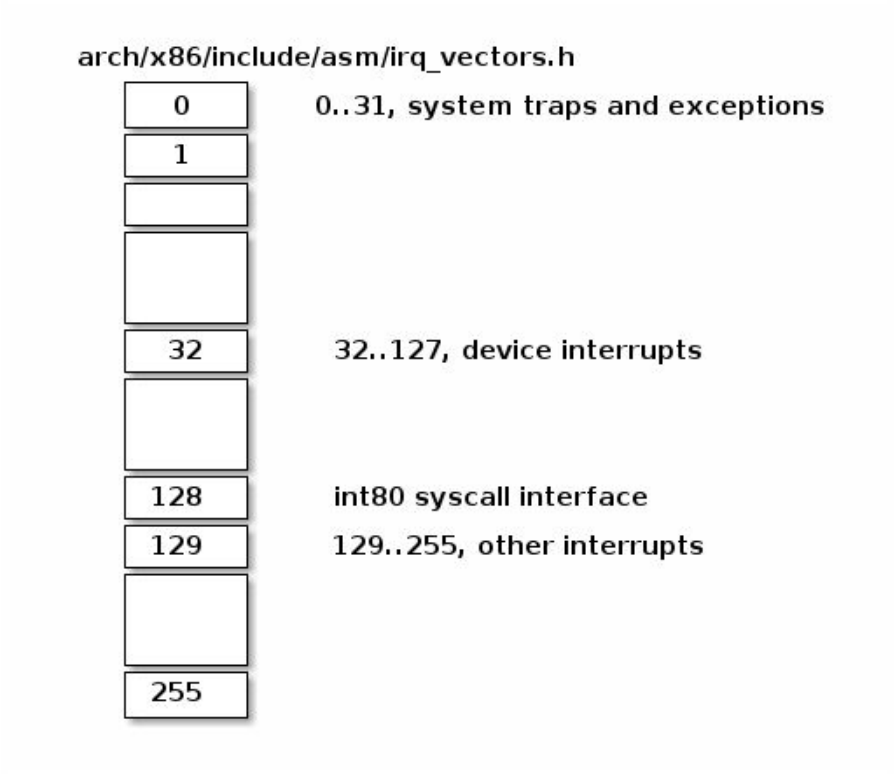
Sudo cat /proc/interrupts

Orden de importancia

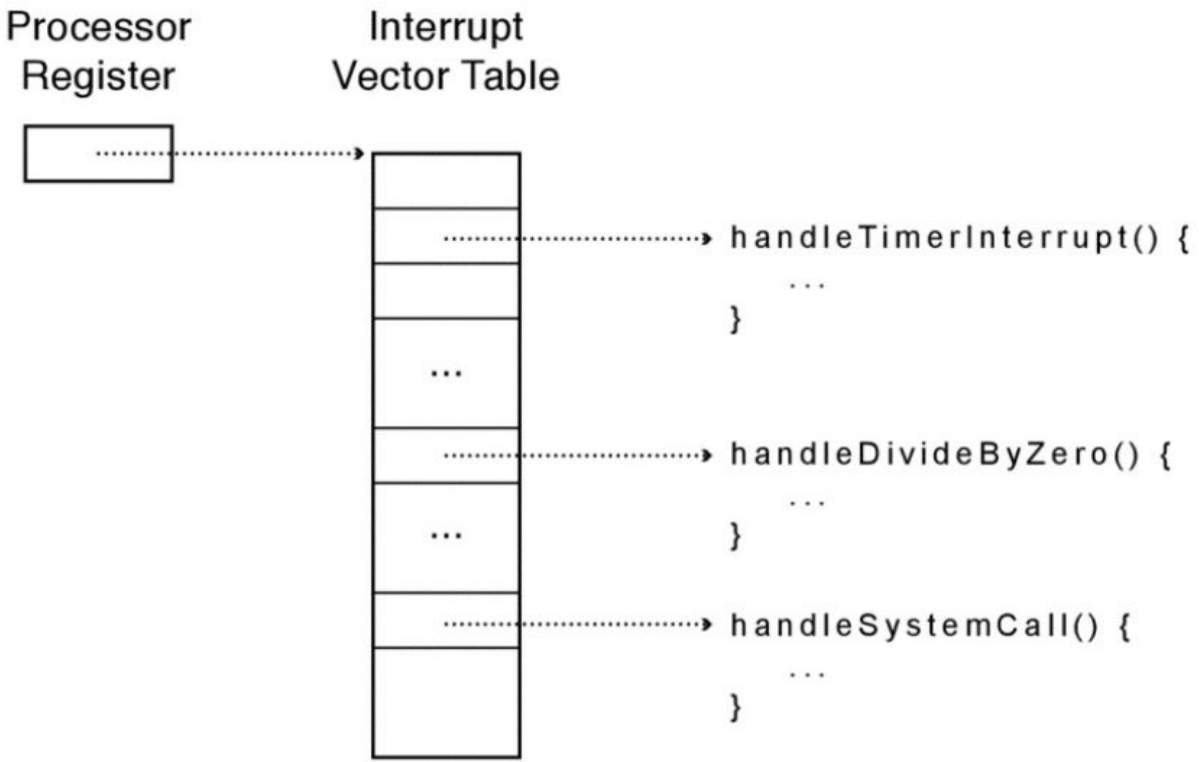
Errores de la Máquina
Timers
Discos
Network devices
Terminales
Interrupciones de Software

Interrupciones

El kernel mantiene una tabla de interrupciones y sus handlers específicos



Interrupciones



Vector de Interrupciones

Se puede imaginar un array que vive en la memoria. Cada entrada en este array se asigna a un número de interrupción. Cada entrada contiene la dirección de una función que la CPU comenzará a ejecutar cuando se reciba esa interrupción junto con algunas opciones, como en qué nivel de privilegio se debe ejecutar la función del controlador de interrupciones.

Aquí hay una foto del manual de la CPU Intel que muestra el diseño de una entrada en este vector en x86:

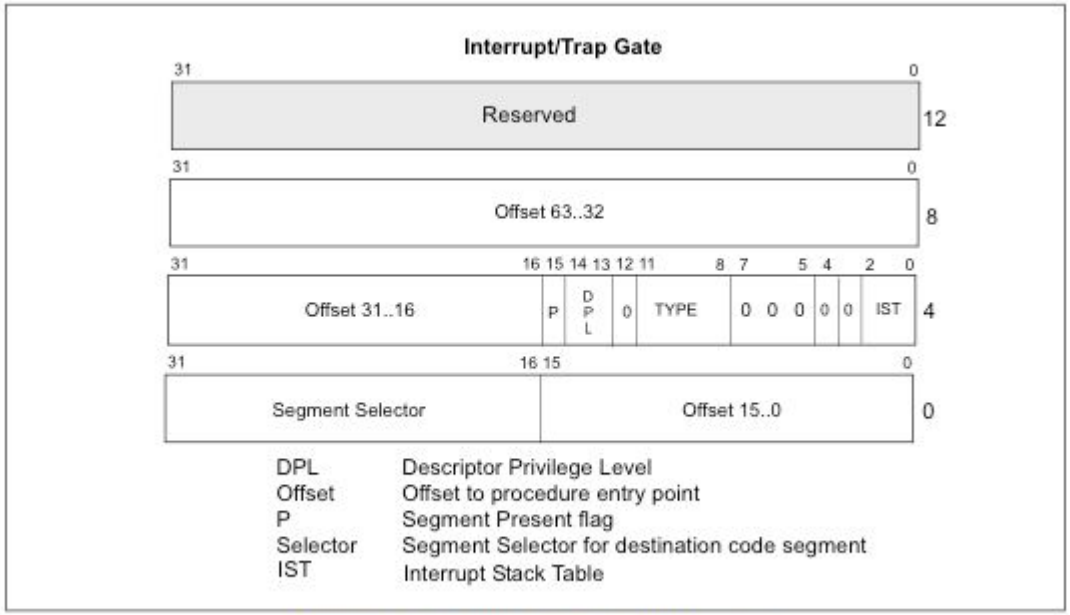
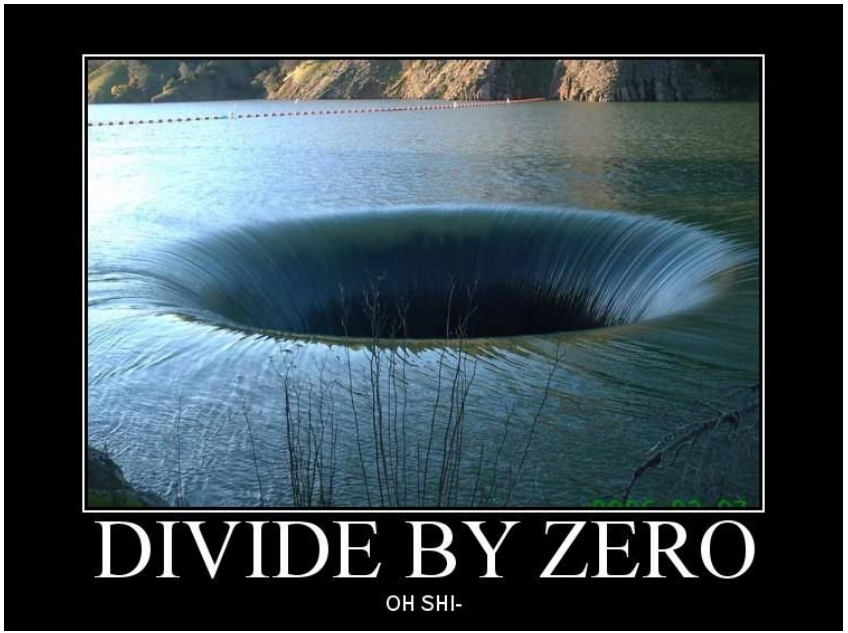


Figure 5-7. 64-Bit IDT Gate Descriptors

Excepciones del Procesador

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

Se suele manejar con un mecanismo similar (o igual) que las interrupciones



Excepciones del Procesador

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

Se suele manejar con un mecanismo similar (o igual) que las interrupciones

```
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);
```

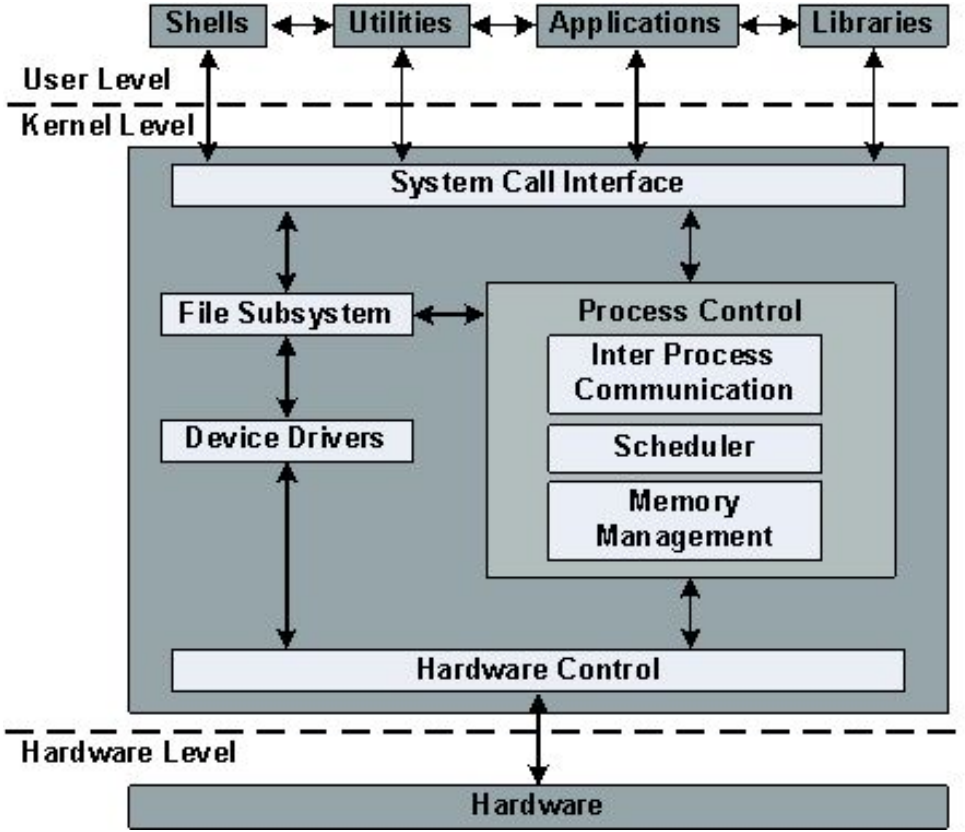
System Calls

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

Una **System Call** es servicio provisto por el kernel que puede ser llamada desde el user level. Básicamente es una función.

$ man 2 syscalls

System Calls



System Calls

¿Como inducimos una transición a una función (system call), pasando de modo Usuario a modo Kernel?

Es lo mismo que induce un dispositivo de I/O al solicitar una interrupción.

Usemos el mecanismo de interrupciones/excepciones/traps para ejecutar una system call!

System Calls

SYSCALL o INT 80 generan una interrupción por software en x86

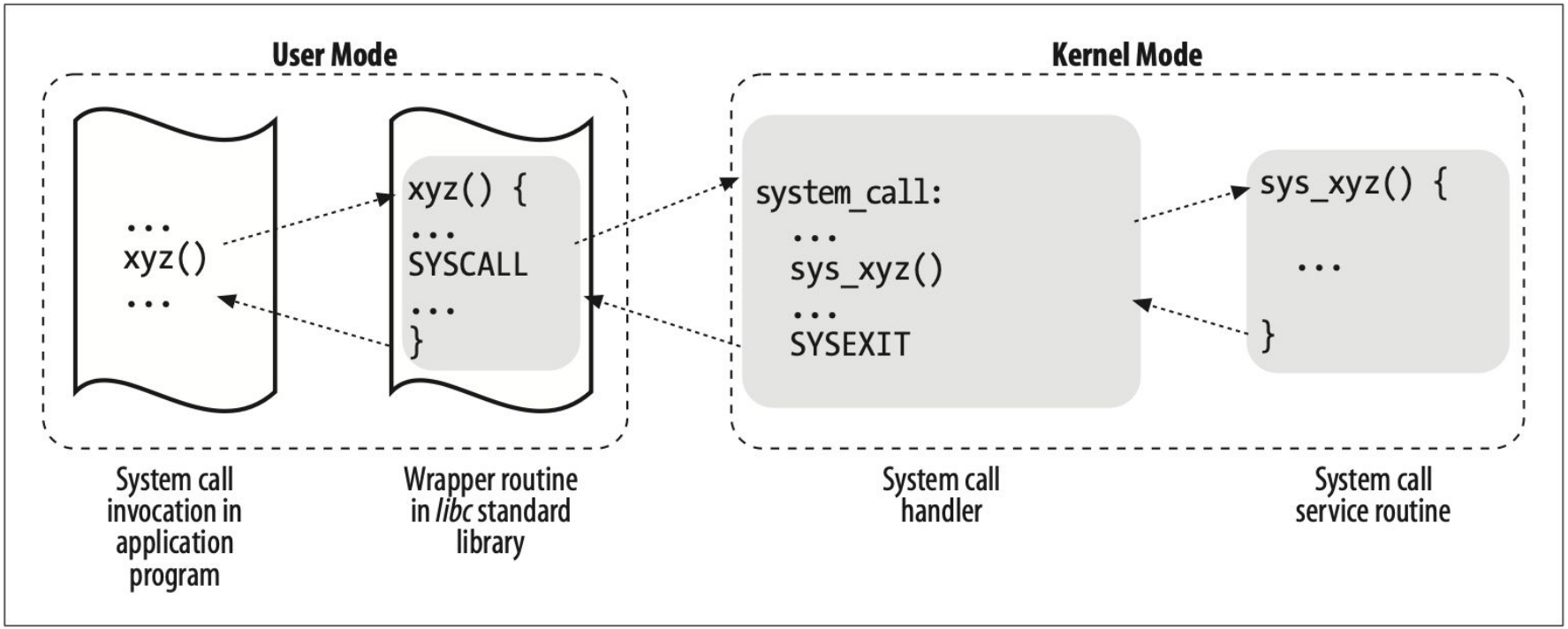


Figure 10-1. Invoking a system call

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:

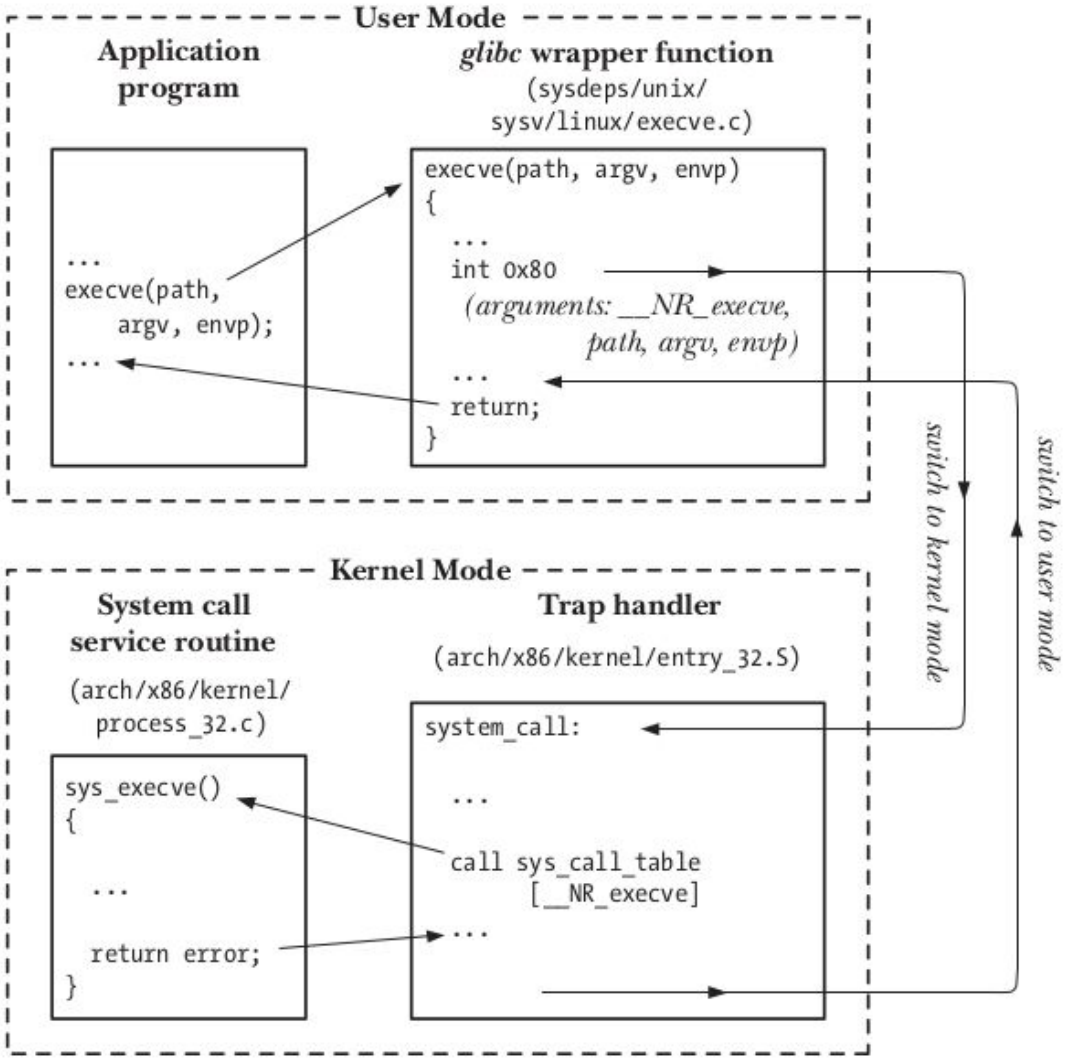
El programa realiza un llamado a una system call mediante la invocación de una **función wrapper** (envoltorio) en la biblioteca de C.

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

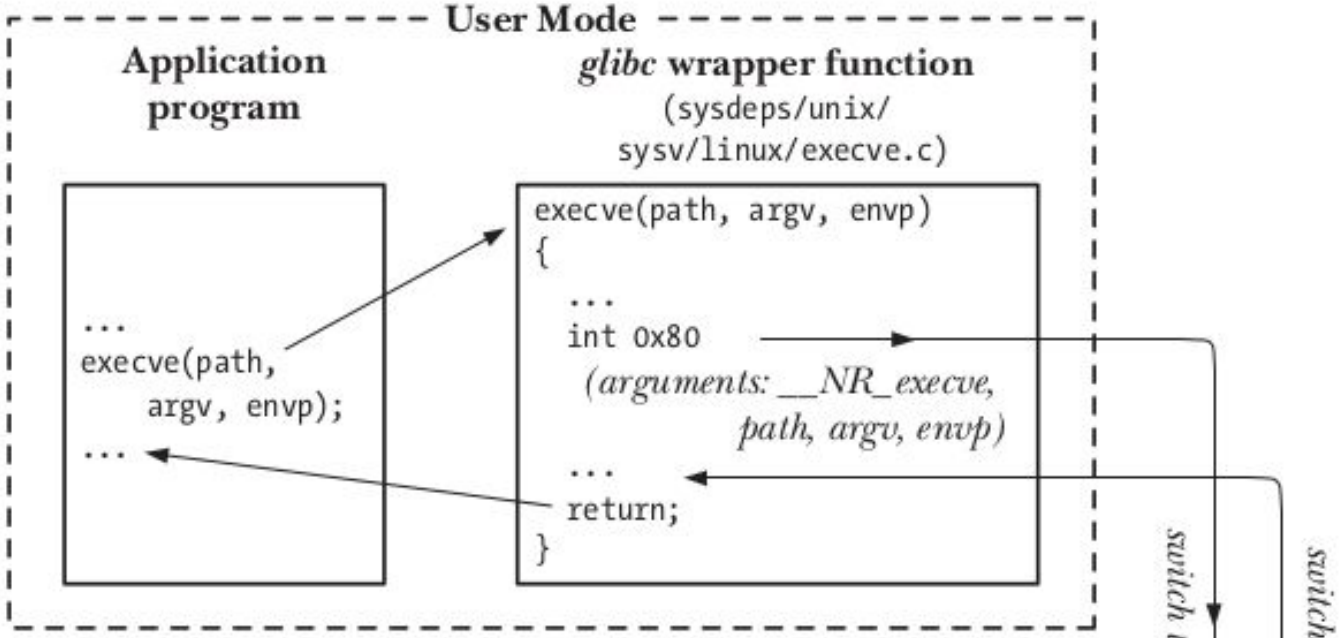
System Calls

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 o SYSCALL), 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.


User Mode
Application program
glibc wrapper function
(sysdeps/unix/sysv/linux/execve.c)
...
execve(path, argv, envp);
...
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)
Trap handler
(arch/x86/kernel/entry_32.S)
sys_execve()
{
...
return error;
}
system_call:
...
call sys_call_table [__NR_execve]
...

System Calls



System Calls

Convención de Syscalls de Linux x86.

```
* 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]
*
```

https://syscalls32.paolostivanin.com/

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

read(2) System Calls Manual *read*(2)

NAME top

read - read from a file descriptor

LIBRARY top

Standard C library (*libc*, *-lc*)

SYNOPSIS top

```
#include <unistd.h>

ssize_t read(int fd, void buf[.count], size_t count);
```

Ejemplo: Invocar la syscall read

```
read(STDIN_FILENO, buffer, buf_len);
```

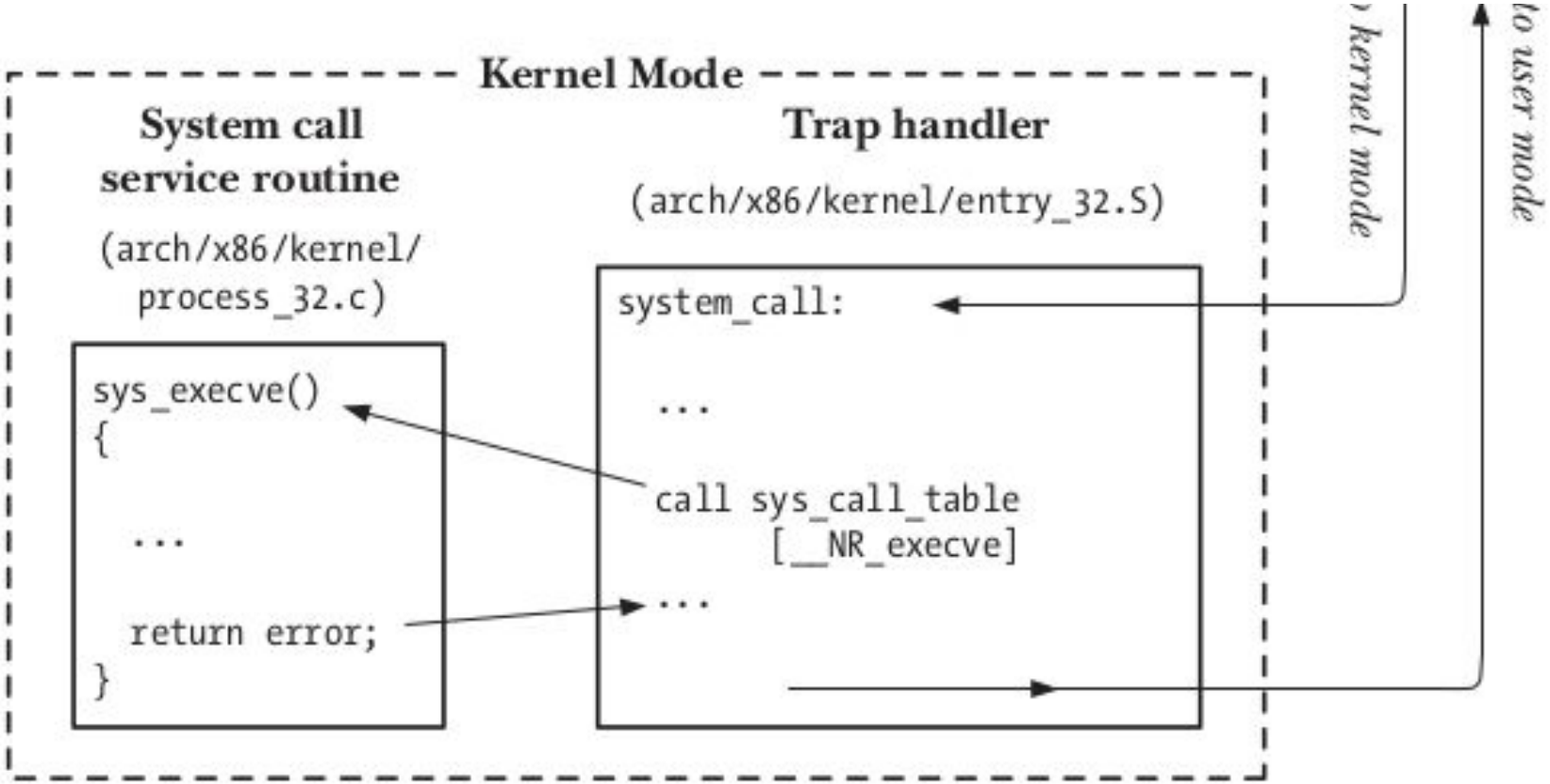
```
mov eax, 3          ; syscall number for sys_read (3)
mov ebx, 0          ; file descriptor (0 for stdin)
lea ecx, [buffer]   ; pointer to the buffer
mov edx, buf_len    ; number of bytes to read
int 0x80            ; make the system call
```

System Calls

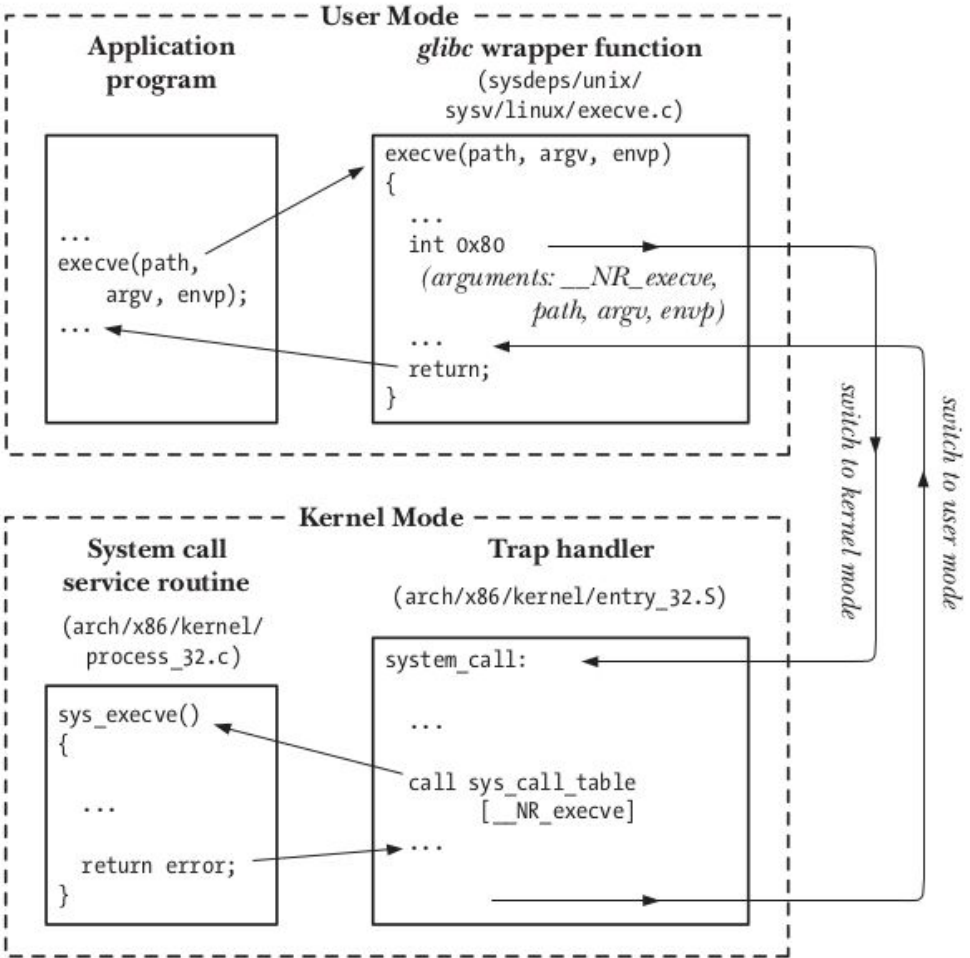
En respuesta al trap, el kernel invoca su propia función llamada syste_call() (arch/i386/entry.s) para manejar esa trap. Este manejador:

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

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

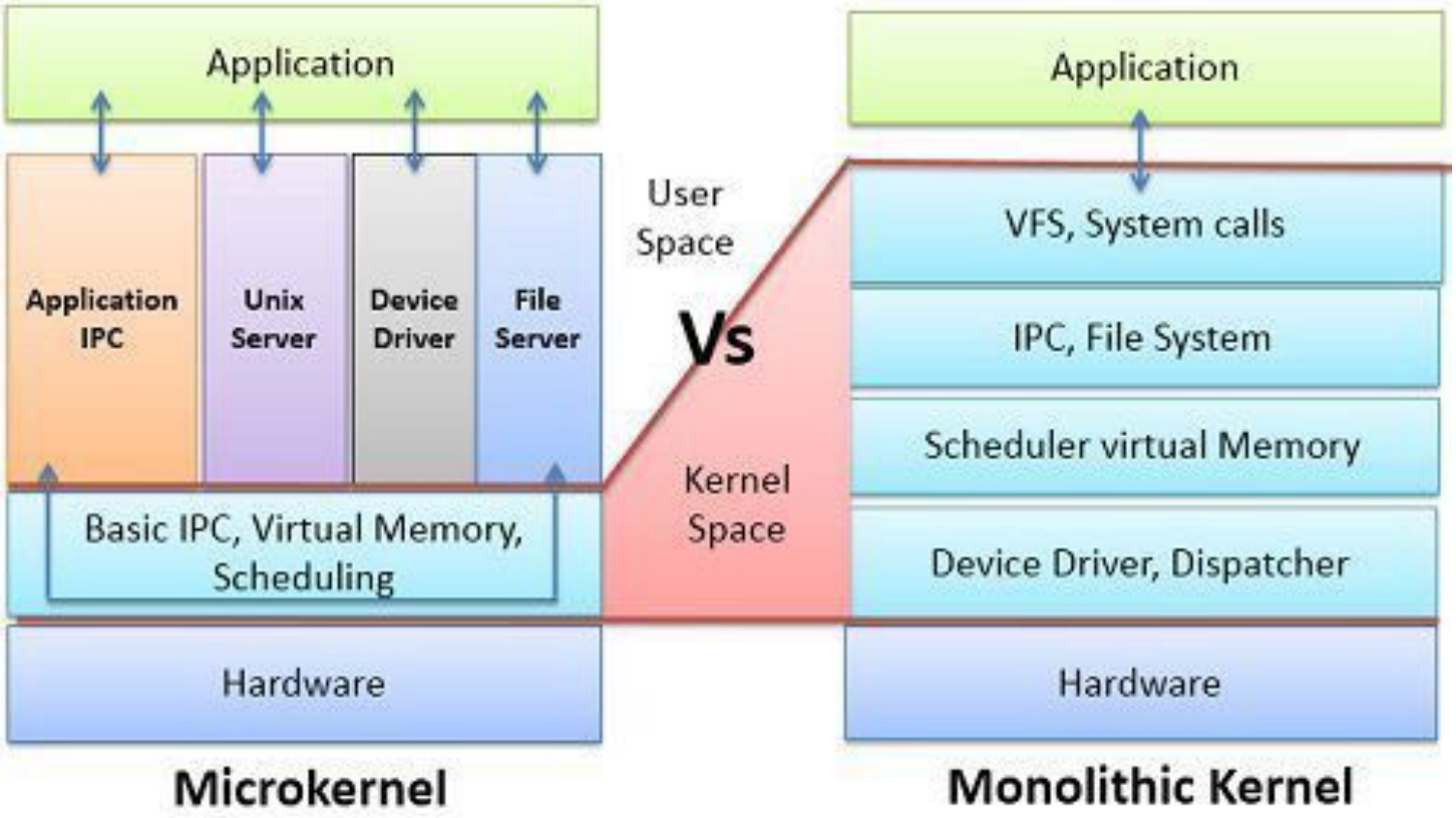
Dahlin Cap 2 hasta 2.4!

Clasificación de kernels

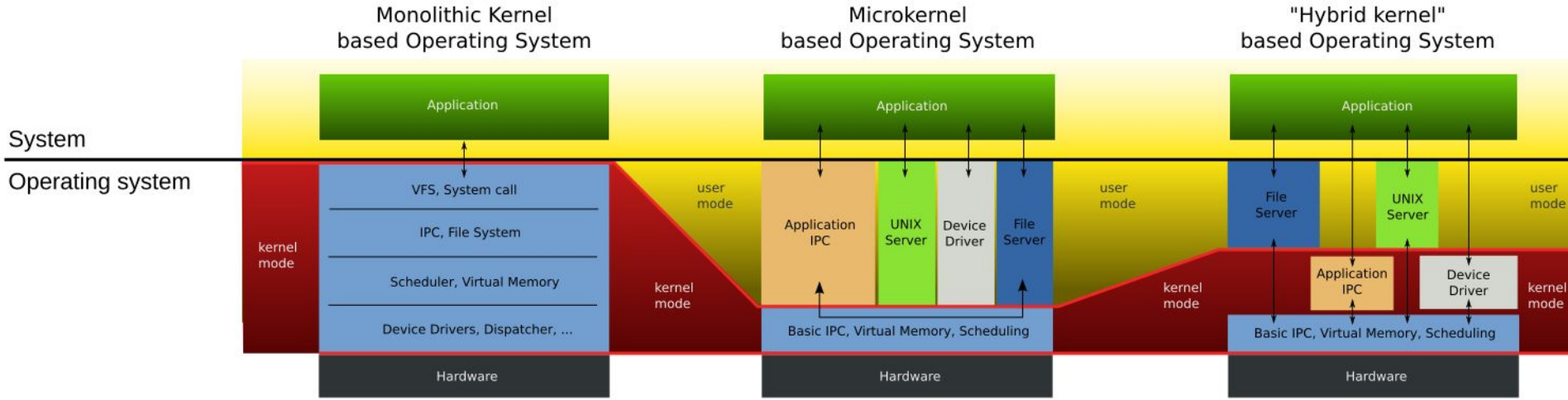
Conceptos clave:

- Monolítico
- Microkernel
- Otros

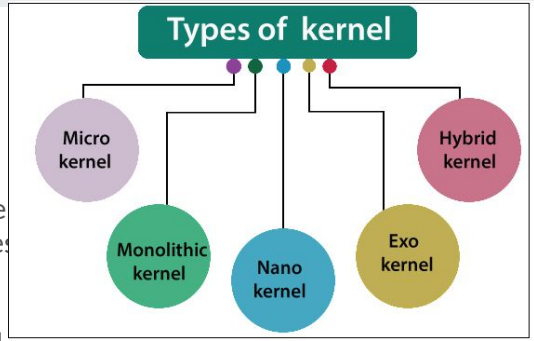
Tipos de Kernel



Tipos de Kernel



Tipos de Kernel



Monolithic Kernel: - In a monolithic kernel, the kernel and operating system, both run in the
used where security is not a major concern. The result of the monolithic kernel is fastly acce
like if a device driver has a bug, then there may be chances of a whole system crash.

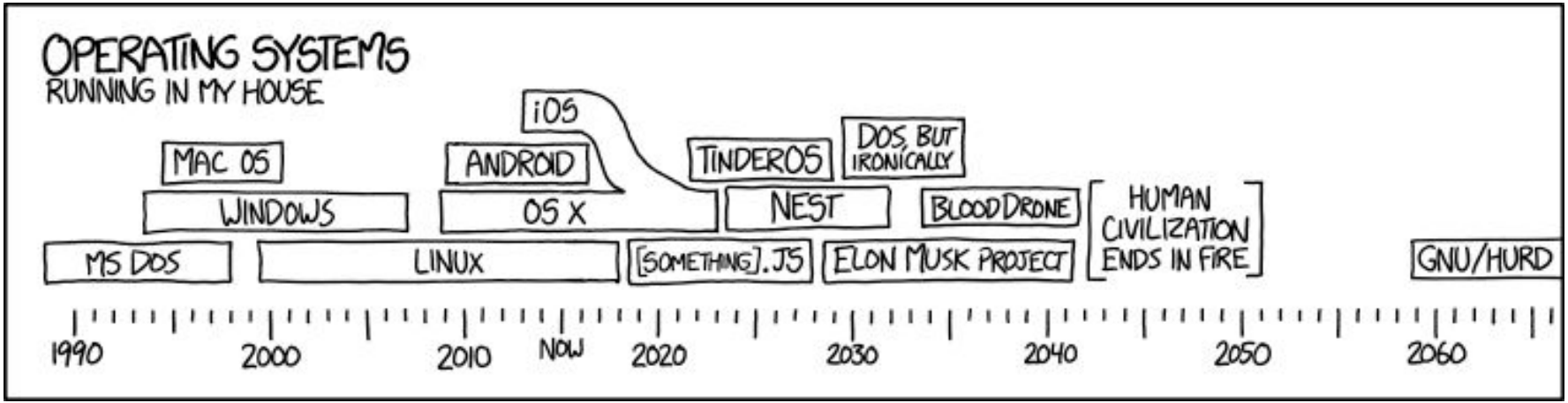
Microkernel: - A Microkernel is the derived version of the monolithic kernel. In microkernel, the kernel itself can do different jobs, and there is no requirement of an additional GUI.

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.

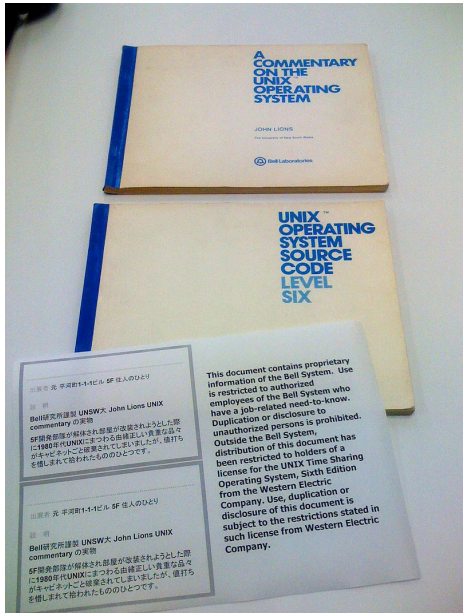
Unix

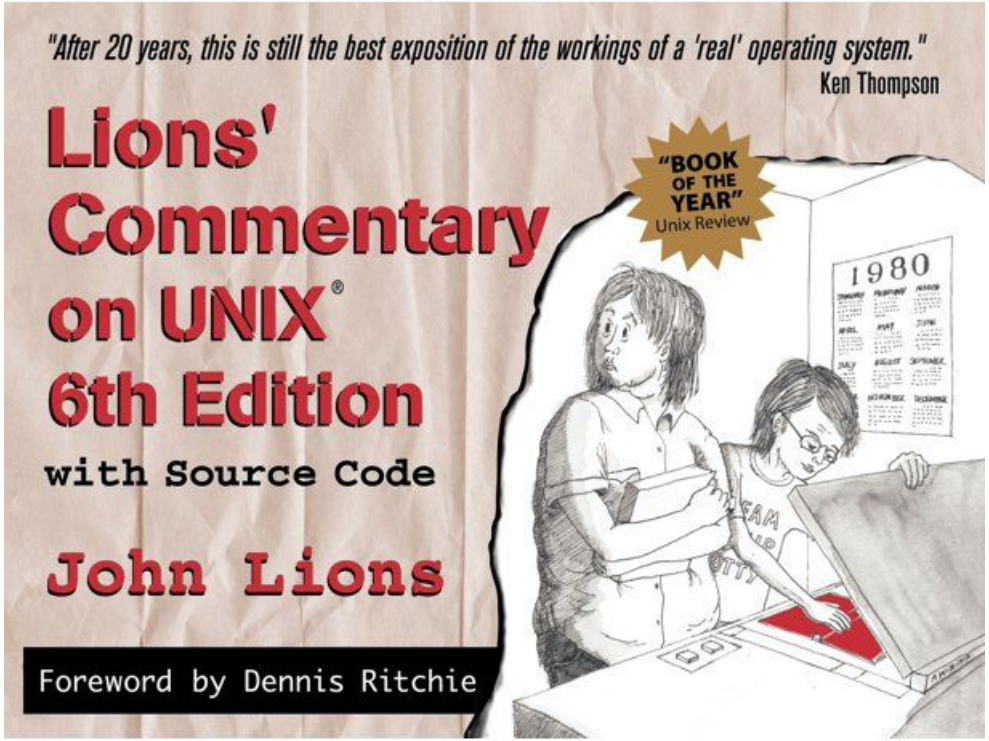

OPERATING SYSTEMS
RUNNING IN MY HOUSE
MS DOS
WINDOWS
MAC OS
LINUX
OS X
ANDROID
iOS
[SOMETHING].JS
NEST
TINDEROS
ELON MUSK PROJECT
DOS, BUT IRONICALLY
BLOOD DRONE
HUMAN CIVILIZATION ENDS IN FIRE
GNU/HURD
1990
2000
2010
NOW
2020
2030
2040
2050
2060

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-/

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 :-(.