

Máquinas Virtuales

PARTE I

Introducción a las máquinas virtuales

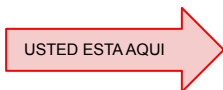
Ejemplos de virtualización

Recurso fisico	Recurso virtualizado
Memoria	Memoria virtual
Procesador	Procesos o threads
Placas de Red	Sockets

Ejemplos de máquinas virtuales

Máquinas virtuales de lenguaje. Eg. Java Virtual Machine, Microsoft Common Language Runtime, V8 Engine (Javascript)

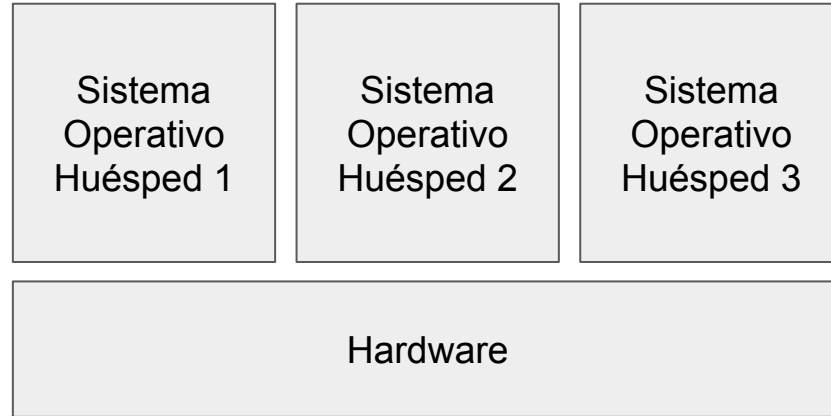
Máquinas virtuales livianas (lightweight). Containers, Docker.



Máquinas virtuales de sistema. Sistema de computación completo, que inclusive corre un sistema operativo sin modificaciones (o casi)

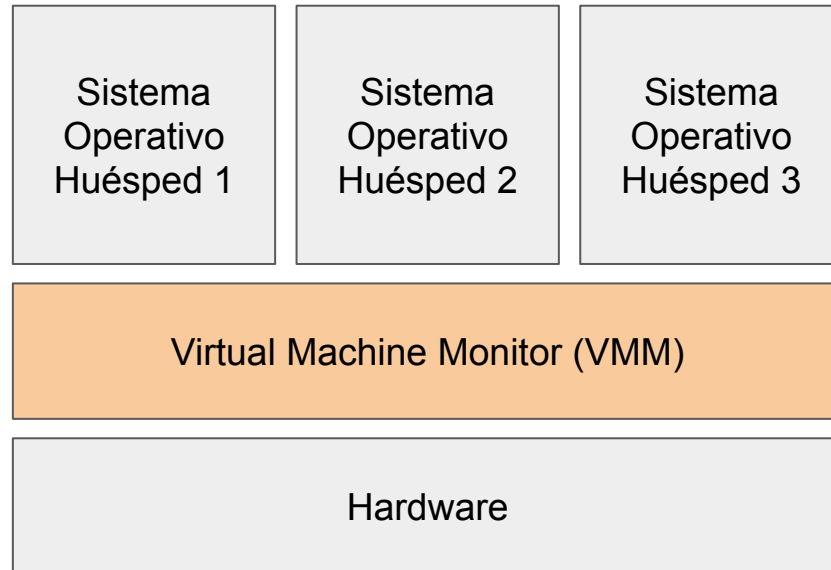
Objetivo

Ejecutar múltiples sistemas operativos sobre un mismo hardware



Objetivo

Ejecutar múltiples sistemas operativos sobre un mismo hardware



¿Por qué queríamos hacer eso?

- Diversidad de Sistemas Operativos
- Consolidación de servidores
- Aprovisionamiento rápido
- Alta disponibilidad
- Seguridad
- Scheduling de recursos distribuidos
- Cloud computing

¿Qué es una máquina virtual?

Una **máquina virtual** es un entorno de computación completo, con sus propias capacidades de procesamiento, memoria y canales de comunicación, aisladas unas de las otras.

Requisitos para la virtualización

- Seguridad.
- Fidelidad.
- Eficiencia.

Requerimientos para la virtualización

Fidelidad (o equivalencia)

El hardware “virtual” es equivalente al hardware real. Un sistema operativo corriendo en una máquina virtual no tiene forma de saber que está corriendo en hardware virtualizado.

Requerimientos para la virtualización

Seguridad

Las máquinas virtuales están aisladas unas de la otra.

ie. cada una no puede acceder a la memoria o recursos de la otra

Requerimientos para la virtualización

Eficiencia

El hardware virtualizado tiene que mostrar, en el peor de los casos, una mínima reducción de velocidad

Tipos de VMM

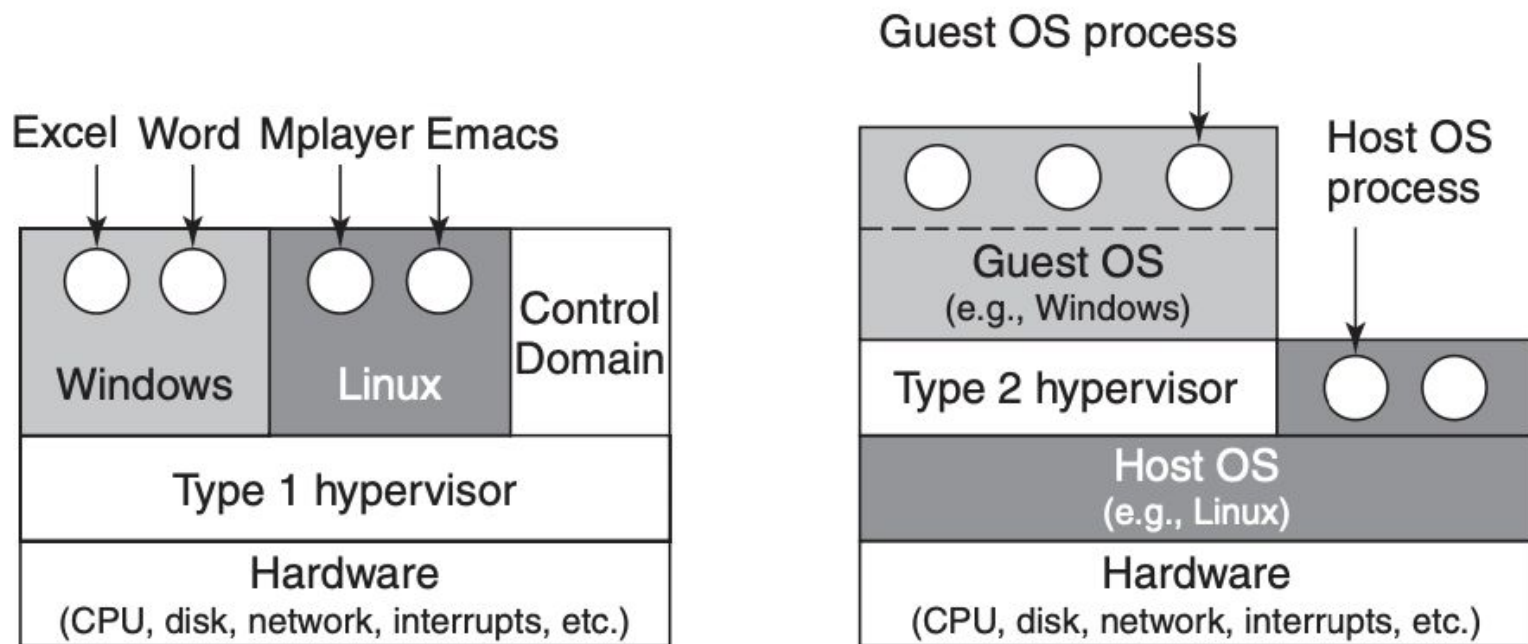


Figure 7-1. Location of type 1 and type 2 hypervisors.

PARTE II

Implementación del VMM

Técnicas de virtualización

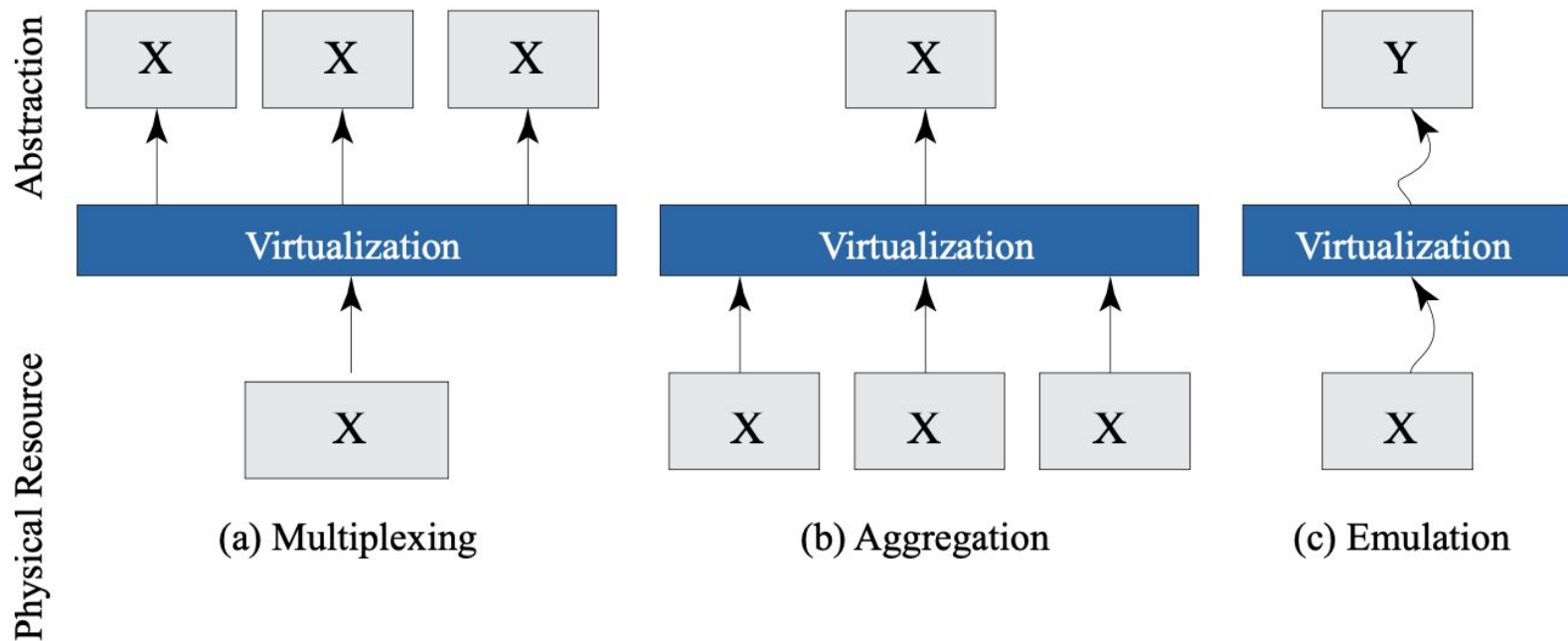


Figure 1.1: Three basic implementations techniques of virtualization. X represents both the physical resource and the virtualized abstraction.

Idea 1: Emular todo

- Emular todas las instrucciones
- Emular toda la memoria
- Emular todos los dispositivos de entrada y salida

Idea 1: Emular todas las instrucciones

- Ejemplo: DOSBox
- Emula una PC vieja con DOS
- Popular para ejecutar juegos en plataformas modernas



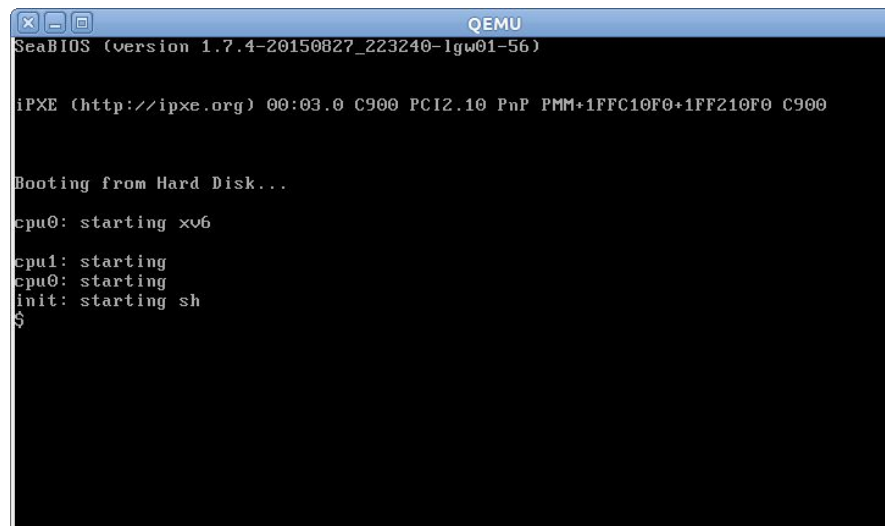
Idea 1: Emular todas las instrucciones

- Ejemplo: Project64
- Emula una consola Nintendo64 sobre arquitecturas intel



Idea 1: Emular todas las instrucciones

- Ejemplo: QEMU
- Emula multiples arquitecturas origen sobre múltiples arquitecturas destino
- Múltiples aceleradores, no solo emulación
- Vease tambien: Bochs (similar a qemu pero solo para emular x86)
- Muy popular en materias de Sistemas Operativos con “TPs desafiantes”



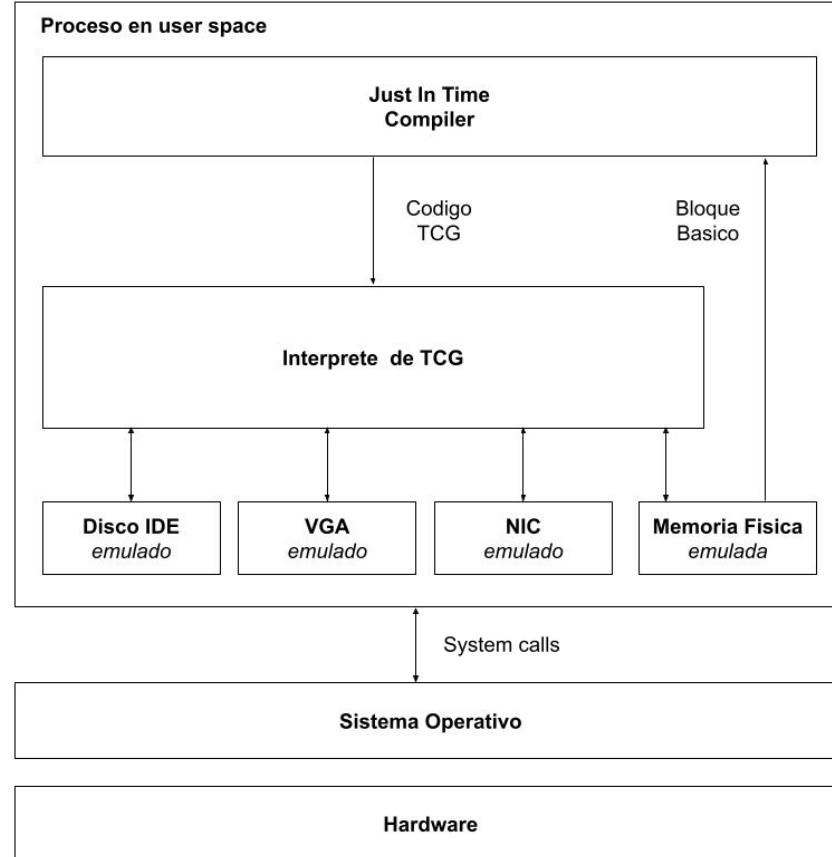
```
QEMU
SeaBIOS (version 1.7.4-20150827_223240-lgw01-56)

iPXE (http://ipxe.org) 00:03.0 C900 PCI2.10 PnP PMM+1FFC10F0+1FF210F0 C900

Booting from Hard Disk...

cpu0: starting xv6
cpu1: starting
cpu0: starting
init: starting sh
$
```

QEMU: Diagrama basico



Idea 1: Emular todas las instrucciones

Cumple con todos los requisitos de virtualización?

- Seguridad
- Equivalencia/Fidelidad
- **Eficiencia**

Problema: Performance. En general el procesador del host debe ser más potente que la máquina emulada.

En particular si se quiere emular la misma arquitectura sobre sí misma se verá una pérdida de performance.

Idea 2: Ejecucion directa

- Las instrucciones se ejecutan directamente en el procesador.
- El VMM multiplexa el CPU entre los guest OSs haciendo cambios de contexto (similar al scheduler con los procesos).
- **Cuando se quiere acceder al hardware real (compartido entre las máquinas virtuales) el VMM toma el control y gestiona el acceso (de alguna forma...)**

Idea 2: Ejecucion directa

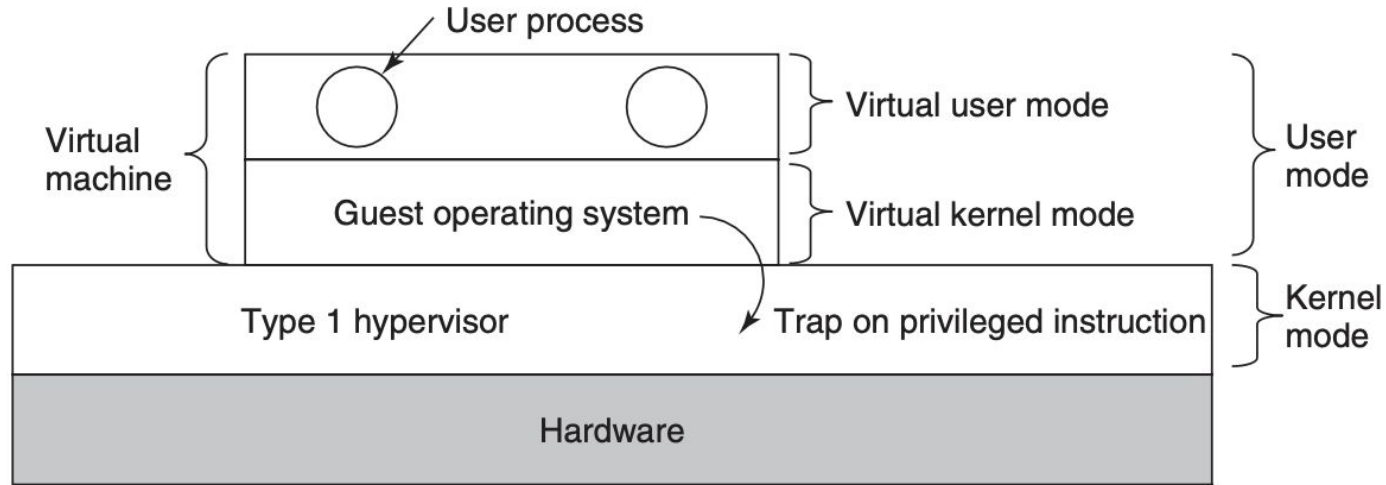


Figure 7-3. When the operating system in a virtual machine executes a kernel-only instruction, it traps to the hypervisor if virtualization technology is present.

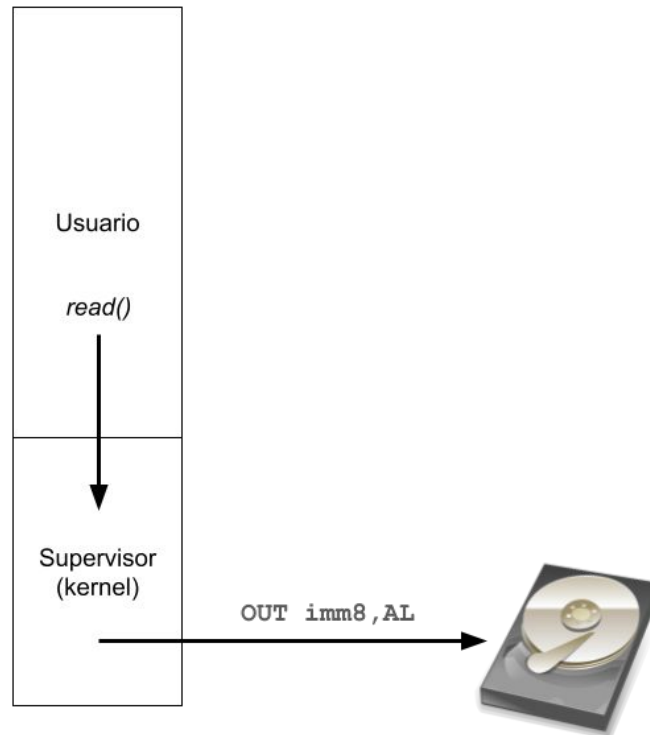
Idea 2: Ejecución directa

Tomemos inspiración de algo que conocemos: **System Calls**

Espacio de direcciones

2 niveles de privilegios:

1. Usuario
2. Supervisor (o kernel)



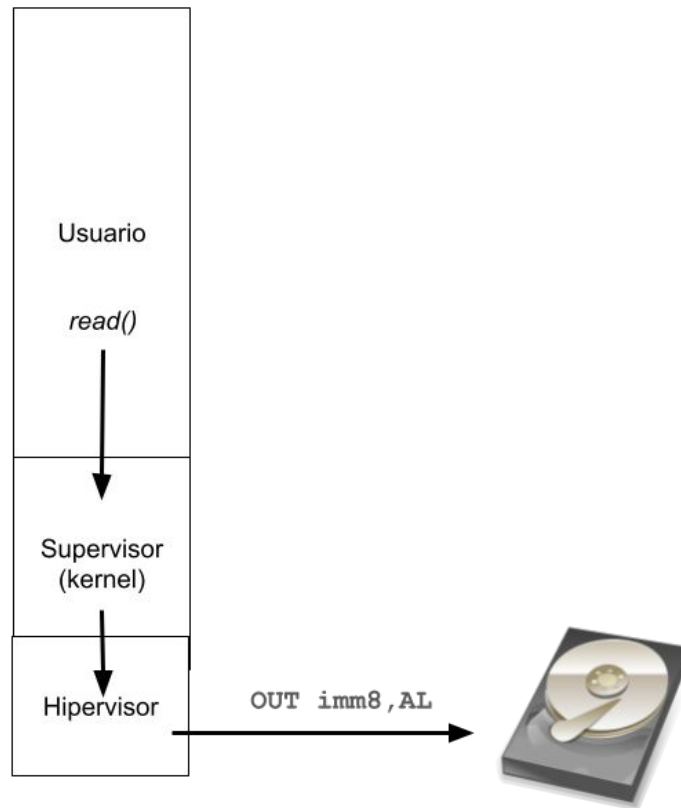
Idea 2: Ejecucion directa

Pongamosle un supervisor al supervisor!

3 modos de ejecucion:

1. Usuario
2. Guest Kernel
3. Hipervisor

Espacio de direcciones



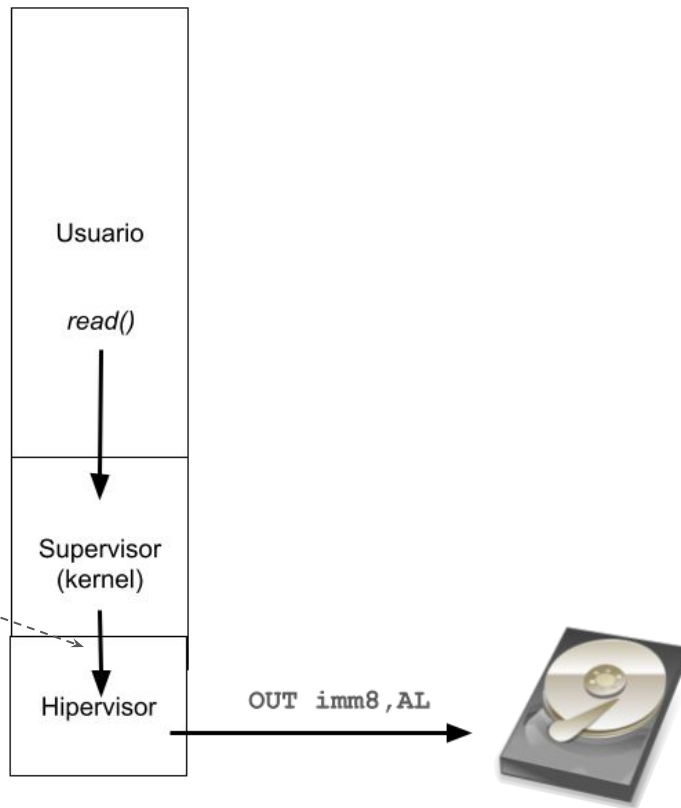
Idea 2: Ejecucion directa

Pongamosle un supervisor al supervisor!

Problema: Cómo se entera el Hipervisor que sus guest OSs están queriendo acceder al hardware?!

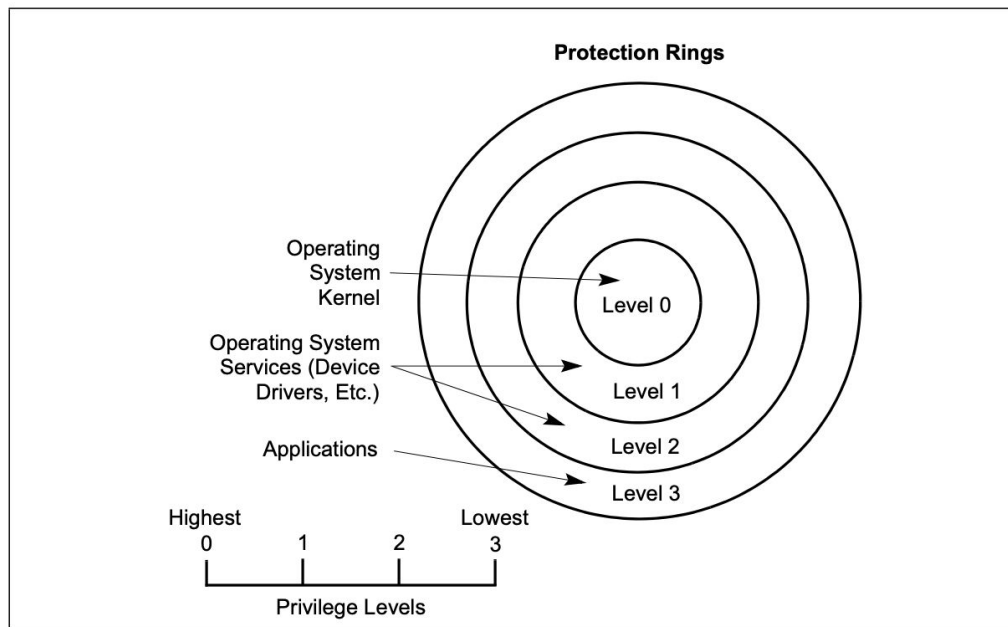
¿Qué nos ofrece el hardware?

Espacio de direcciones



Digresión: Modo protegido en x86

Modo protegido en x86 (simplificado)



- El procesador siempre se encuentra en un **CPL (current privilege level)**
- Los recursos del CPU tienen un nivel requerido. Múltiples mecanismos. Eg.
 - Segmentos: DPL (en el segment descriptor)
 - Puertos de IO: IOPL (en el registro EFLAGS)
 - Páginas: User/Supervisor flag (en entradas de la page table)
- Tratar de acceder a un recurso con un CPL insuficiente genera una **General Protection Fault**

Modo protegido en x86 (simplificado)

Ejemplo: Instruccion OUT

OUT — Output to Port : Transfers a data byte or data word from the register (AL, AX, or EAX) given as the second operand to the output port numbered by the first operand.

Eg.

```
OUT imm8,AL
```

Modo protegido en x86 (simplificado)

Ejemplo: Instruccion OUT - Operacion

```
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** el chequeo de permisos, en **AZUL** la operacion en si misma

Referencias:

PE = Modo protegido activado

CPL = current privilege level

IOPL = I/O privilege level

#GP(0) = General Protection Fault codigo 0

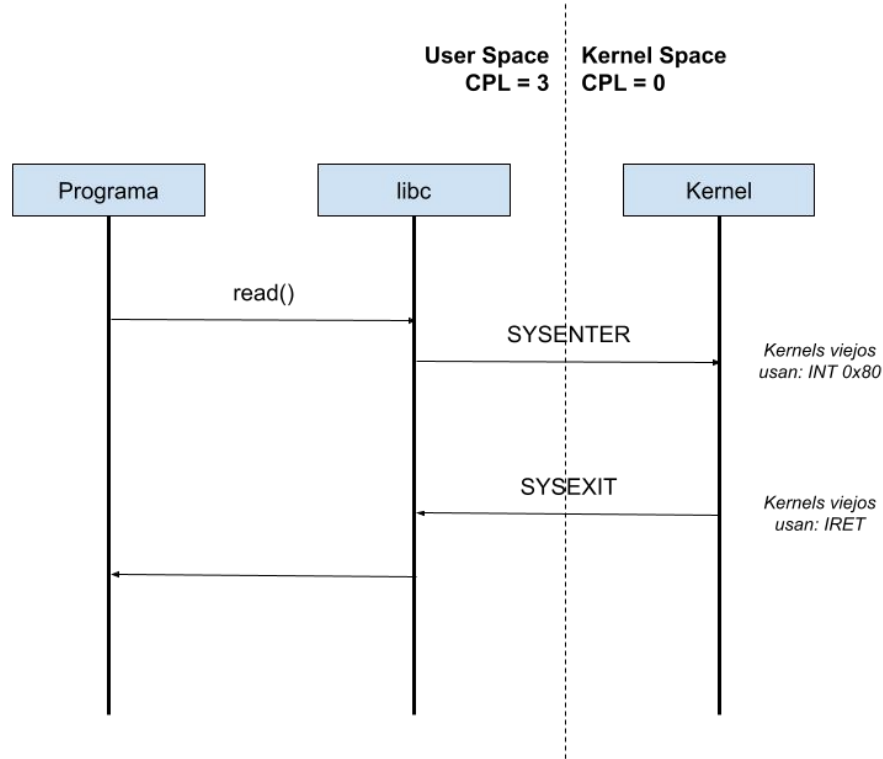
VM = Virtual 8086 mode

I-O-Permission = True si la "tarea" actual tiene permisos especificos para ese puerto

Modo protegido en x86 (simplificado)

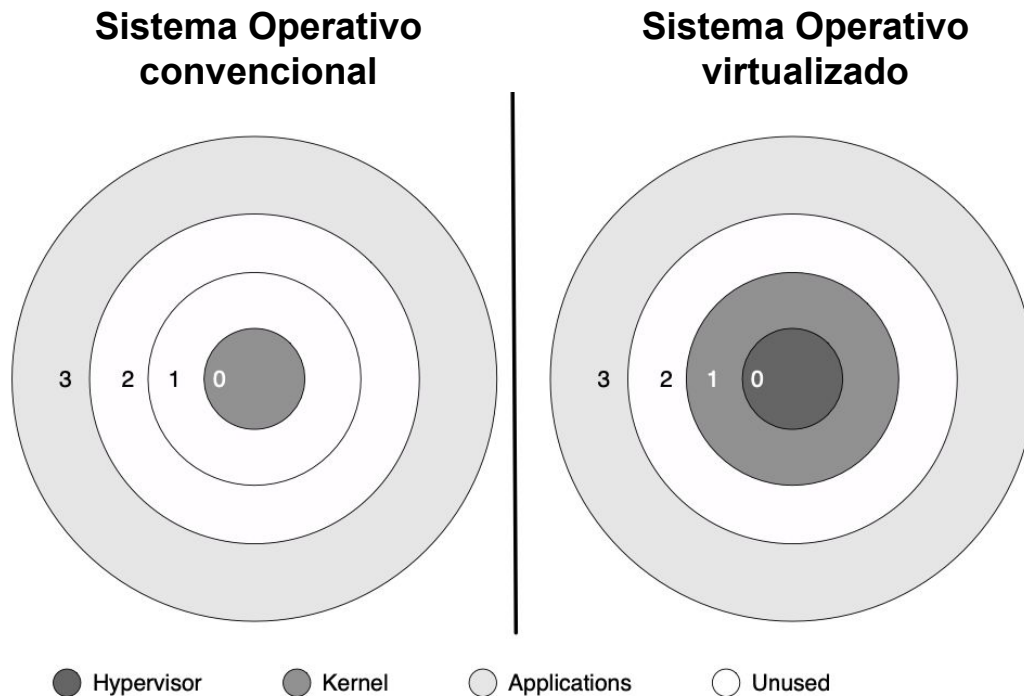
Para pasar de Modo Usuario (CPL 3) a Modo Supervisor (CPL 0) mediante una System Call hay un protocolo bien definido que los programas ejecutan

Modo protegido en x86 (simplificado)



Idea 2: Ejecución directa

Usemos los anillos que están disponibles



Modo protegido en x86 (simplificado)

Como pasar del CPL 1 al CPL 0 sin que el Sistema Operativo Huesped se entere?

Trucazo:

Aprovechemos que el CPU salta solo de anillo cuando hay una General Protection Exception.

Dejemos que el Guest se choque solo contra la GP fault y dejemos que la VMM lo atrape

Idea 2: Ejecución directa

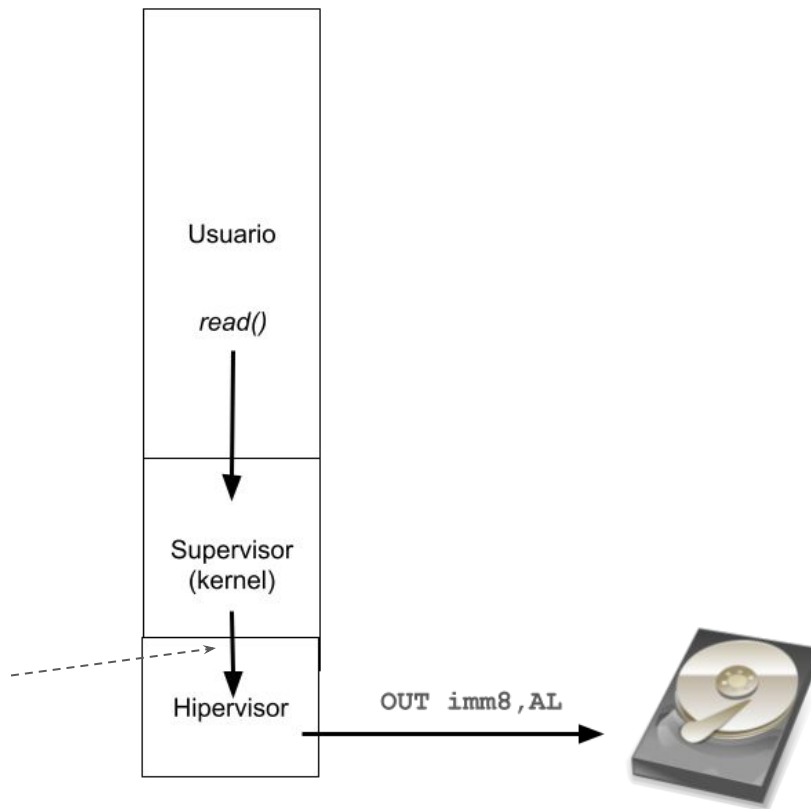
Pongamosle un supervisor al supervisor!

Trap and Emulate.

1. Aprovechar que el Guest Kernel ahora no está en el anillo más privilegiado.
2. Capturar las **General Protection Fault** en el Hypervisor
3. Emular la instrucción privilegiada y devolver el control al guest kernel.

General Protection Fault

Espacio de direcciones



Idea 2: Ejecucion directa

Hemos solucionado el problema de la virtualización?

NO

(Instruccion POPF 😞)

Instruccion POPF

POPF/POPFD — Pop Stack into FLAGS or EFLAGS Register

POPF/POPFD pops the word or doubleword on the top of the stack and stores the value in the flags register.

[...]

If a POPF instruction is executed with insufficient privilege, an exception does not occur, but the privileged bits do not change.

PARTE III

Virtualizar lo invirtualizable

Requisitos para construir un VMM

Teorema: Un VMM se puede construir si el conjunto de instrucciones sensibles es un subconjunto de instrucciones privilegiadas

Instrucciones sensibles: Instrucciones que se ejecutan diferente si están en modo usuario o kernel. Eg. I/O, modificar MMU, etc

Instrucciones privilegiadas: Instrucciones que causan traps en modo usuario pero no en modo supervisor

Requisitos para construir un VMM

Teorema (en castellano): Si tratas de hacer algo en user mode que no deberías, el sistema va a generar un trap (excepción por hardware). Si eso no pasa en todos los casos, vas a tener problemas para construir un VMM.

Cumple x86 con los requisitos para ser virtualizada?

Table 2.2: List of sensitive, unprivileged x86 instructions

Group	Instructions
Access to interrupt flag	pushf, popf, iret
Visibility into segment descriptors	lar, verr, verw, lsl
Segment manipulation instructions	pop <seg>, push <seg>, mov <seg>
Read-only access to privileged state	sgdt, sldt, sidt, smsw
Interrupt and gate instructions	fcall, longjump, retfar, str, int <n>

Tres estrategias para virtualizar x86

- Traducción binaria
- Paravirtualización
- Virtualización asistida por hardware

Traducción binaria

Estrategia: traducción en tiempo de ejecución de las instrucciones sensibles

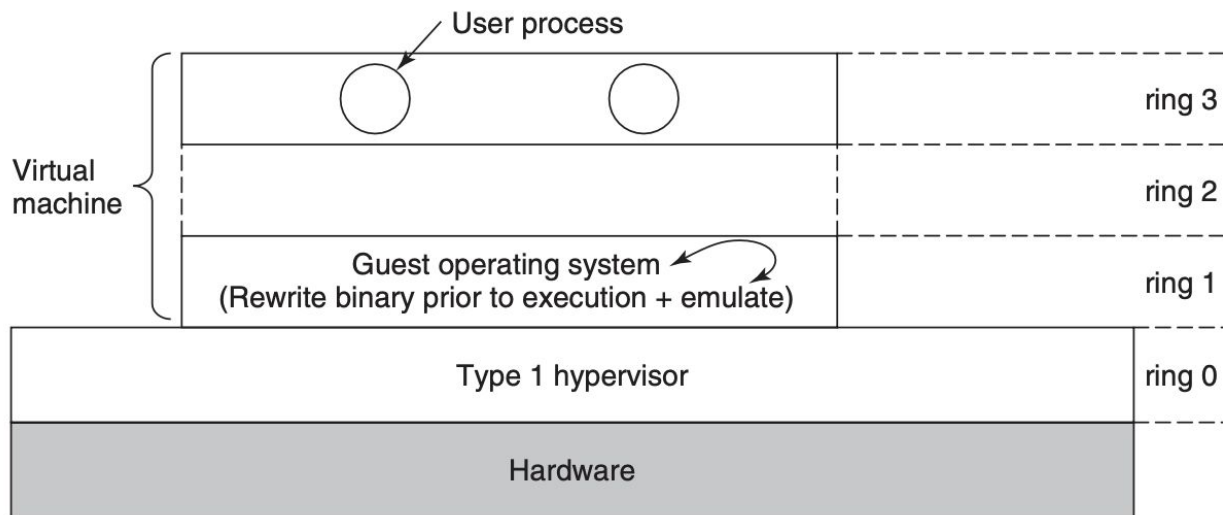


Figure 7-4. The binary translator rewrites the guest operating system running in ring 1, while the hypervisor runs in ring 0.

Paravirtualización

Estrategia: hacer trampa. El OS sabe que está corriendo sobre un VMM, sabe comunicarse con él, y no trata de hacer cosas “sensibles” directamente.

User a Kernel -> System Call
Kernel a Hypervisor -> Hypercall

Nota: Esta idea existe desde 1972 y fue revivida a principios de los 2000 con Xen

Virtualización asistida por hardware

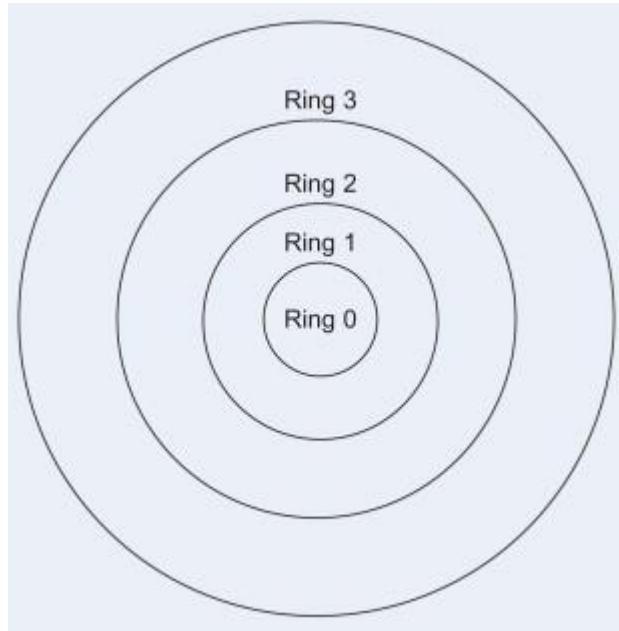
Estrategia: cambios a la arquitectura del procesador para (con algunas complicaciones) cumplir con el teorema de virtualización.

Dos conjuntos de extensiones, incompatibles entre sí (pero parecidos):

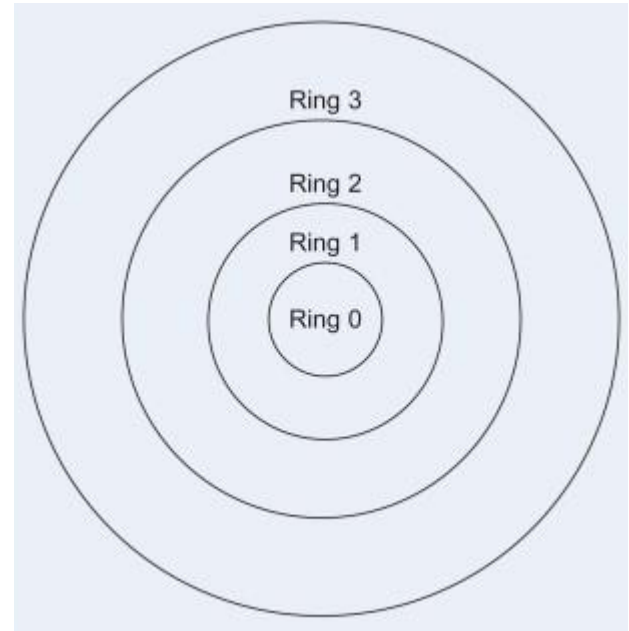
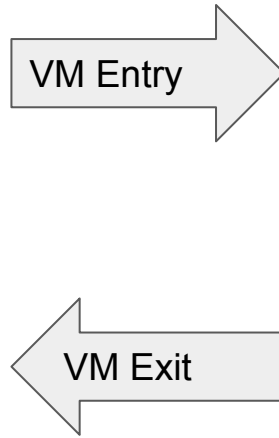
Intel: VT (Virtualization Technology)

AMD: SVM (Secure Virtual Machine)

Virtualización asistida por hardware



root mode



non-root mode

Virtualización asistida por hardware

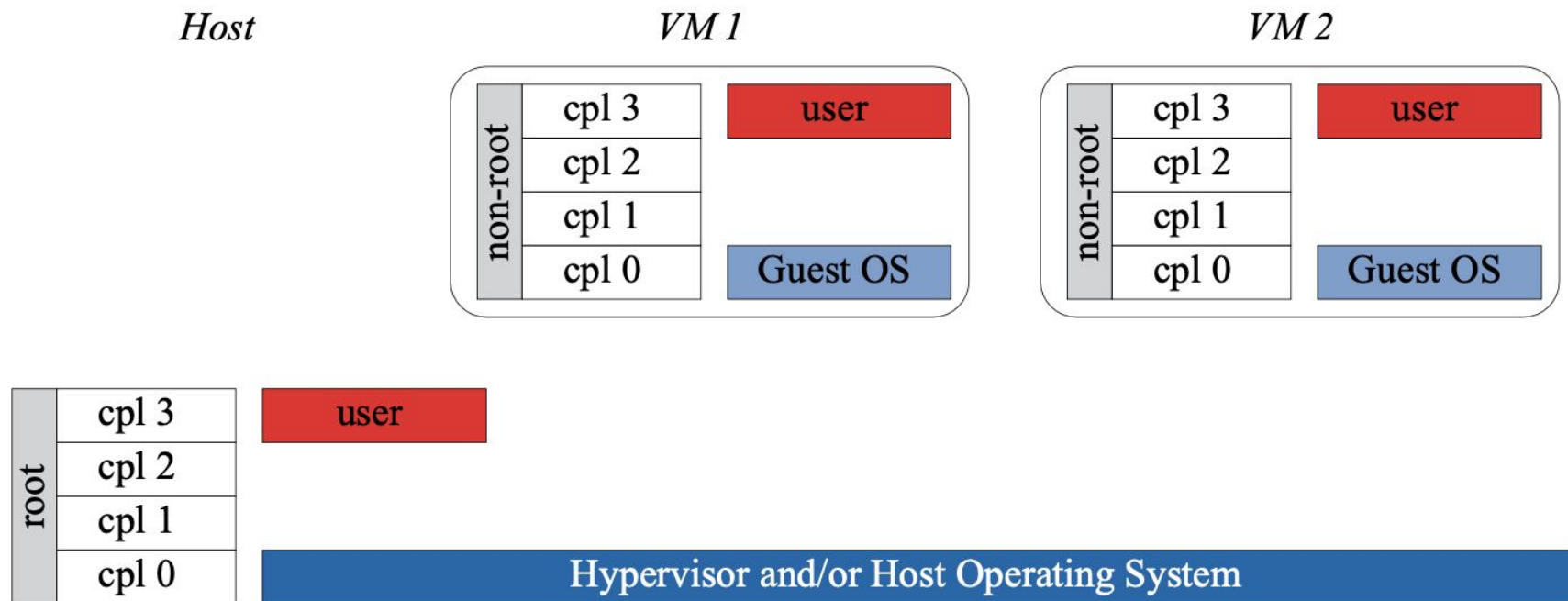


Figure 4.1: Standard use by hypervisors of VT-x root and non-root modes.

Virtualización asistida por hardware

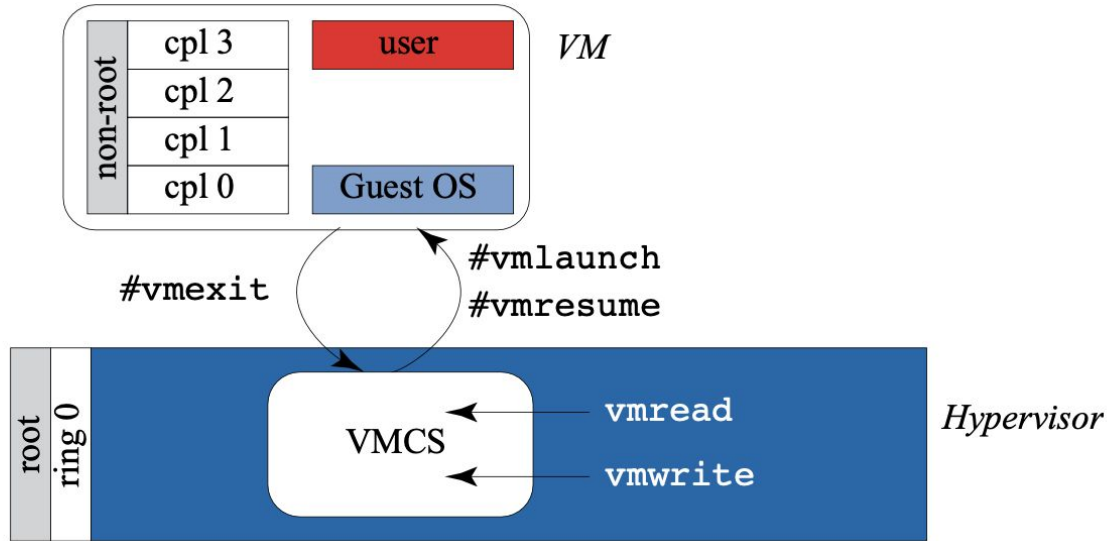


Figure 4.2: VT-x transitions and control structures.

VMCS: Virtual Machine Control Structure

Es una estructura especial que contiene el estado de la máquina virtual y se guarda en memoria física.

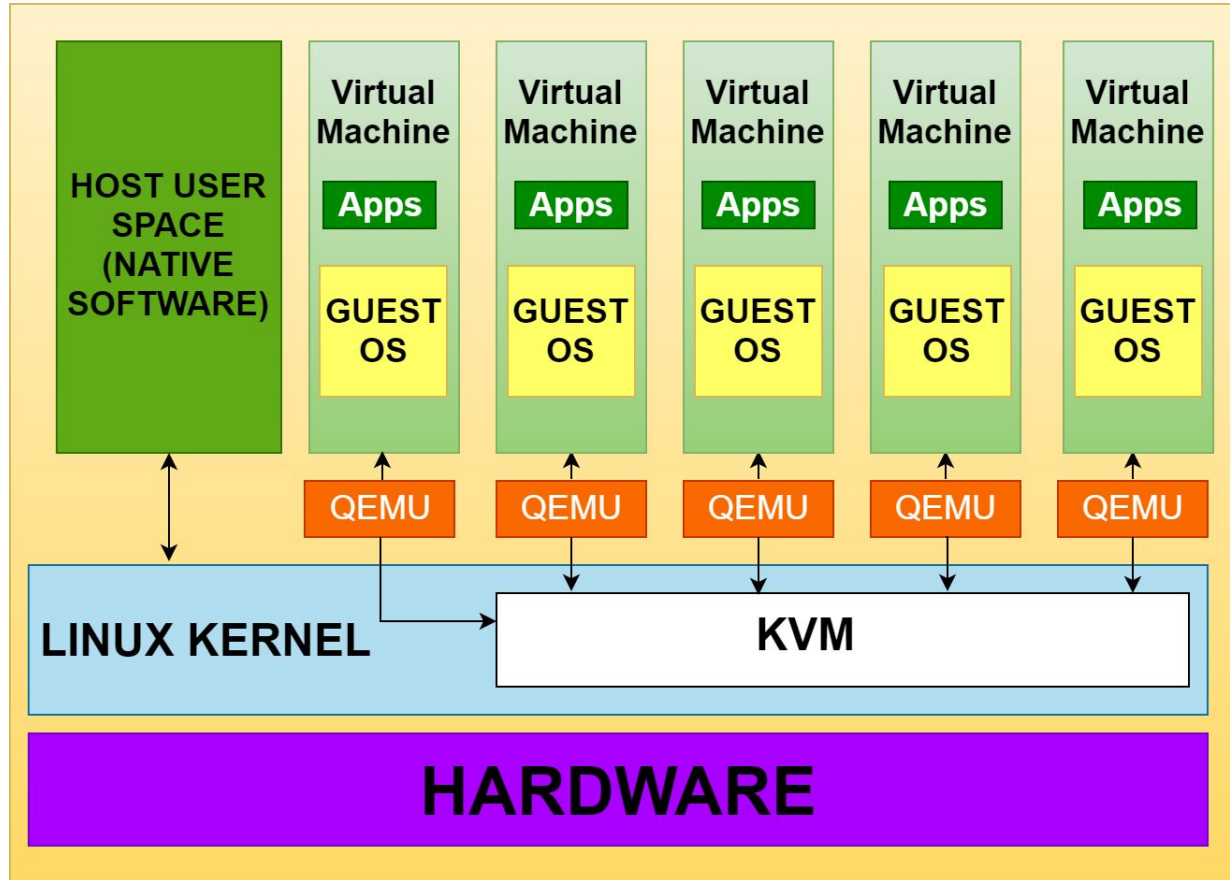
vmlaunch y **vmresume** son instrucciones privilegiadas

Virtualización asistida por hardware

Table 4.1: Categories VT-x exit codes

Category	Exit Reason	Description
Exception	0	Any guest instruction that causes an exception
Interrupt	1	The exit is due to an external I/O interrupt
Triple fault	2	Reset condition (bad)
Interrupt window	7	The guest can now handle a pending guest interrupt
Legacy emulation	9	Instruction is not implemented in non-root mode; software expected to provide backward compatibility, e.g., <code>task switch</code>
Root-mode Sensitive	11-17, 28-29, 31-32, 46-47:	x86 privileged or sensitive instructions: <code>getsec</code> , <code>hlt</code> , <code>invd</code> , <code>invlpg</code> , <code>rdpmc</code> , <code>rdtsc</code> , <code>rsm</code> , <code>mov-cr</code> , <code>mov-dr</code> , <code>rdmsr</code> , <code>wrmsr</code> , <code>monitor</code> , <code>pause</code> , <code>lgdt</code> , <code>lidt</code> , <code>sgdt</code> , <code>sidt</code> , <code>lldt</code> , <code>ltr</code> , <code>sldt</code>
Hypercall	18	<code>vmcall</code> : Explicit transition from non-root to root mode
VT-x new	19-27, 50, 53	ISA extensions to control non-root execution: <code>invept</code> , <code>invvpid</code> , <code>vmclear</code> , <code>vmlaunch</code> , <code>vmpttrld</code> , <code>vmptrst</code> , <code>vmreas</code> , <code>vmresume</code> , <code>vmwrite</code> , <code>vmxoff</code> , <code>vmxon</code>
I/O	30	Legacy I/O instructions
EPT	48-49	EPT violations and misconfigurations

Virtualización moderna en Linux: KVM



Virtualización moderna en Linux: KVM

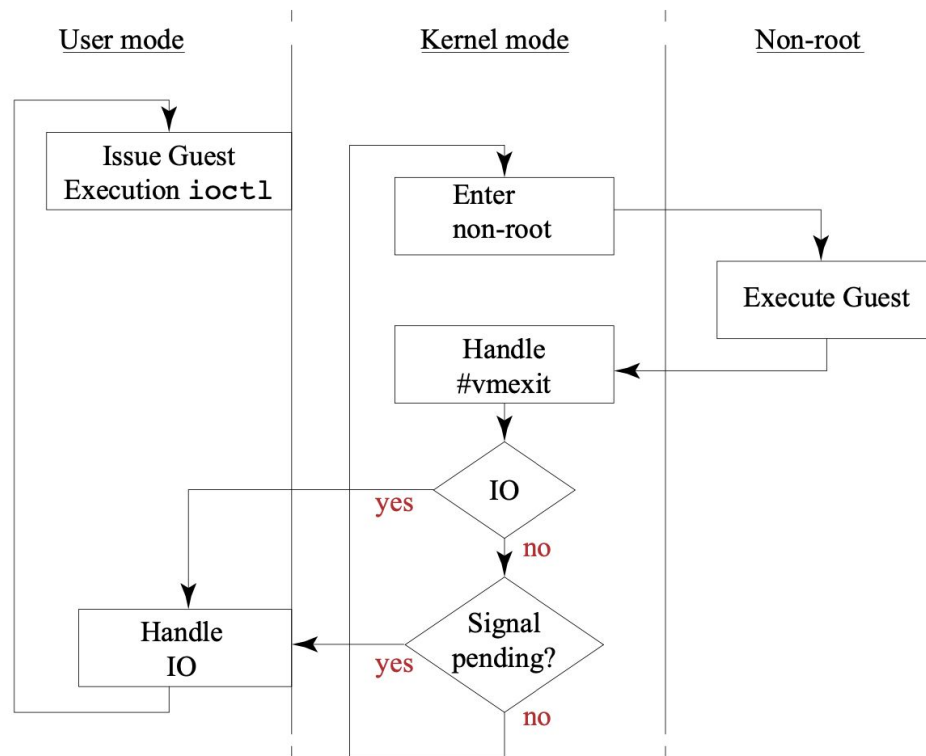


Figure 4.4: KVM Virtual Machine Execution Loop, adapted from [113].

Bibliografia

- ***Hardware and Software Support for Virtualization* Edouard Bugnion, Jason Nieh, and Dan Tsafir**
La principal referencia para esta clase. Además tiene buenas referencias a otras bibliografías.
Recomiendo esta serie, también hay uno sobre memoria virtual interesante *Architectural and Operating System Support for Virtual Memory* - Bhattacharjee, Daniel Lustig
- ***Modern Operating Systems - Fourth Edition* Andrew S. Tanenbaum Herbert Bos - Capítulo 7** El
Tanenbaum en general no está muy bueno por ser muy opinado y tener mucho texto, pero el capítulo de virtualización está bueno.
- ***Principles of Computer System Design* - Jerome H. Saltzer, M. Frans Kaashoek - Capítulo 5.8**
- ***The definitive guide to the Xen hypervisor* / David Chisnall.** Principalmente por motivos históricos, es sobre la primera versión del Xen que funcionaba paravirtualizada