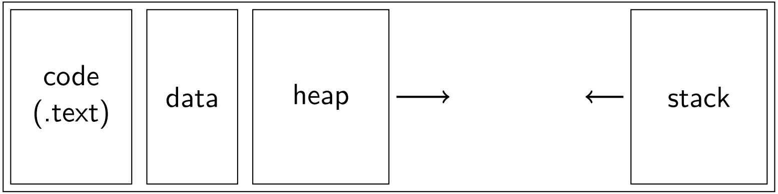
Virtualisation

Johan Montelius

KTH

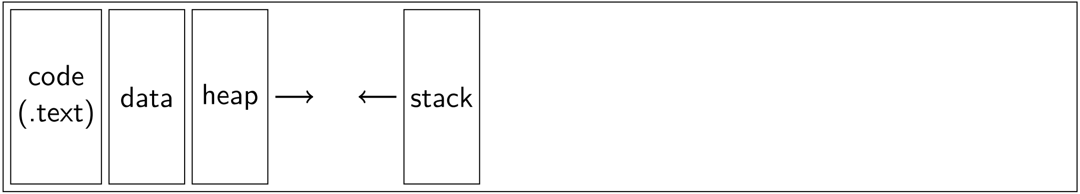
2021

the process

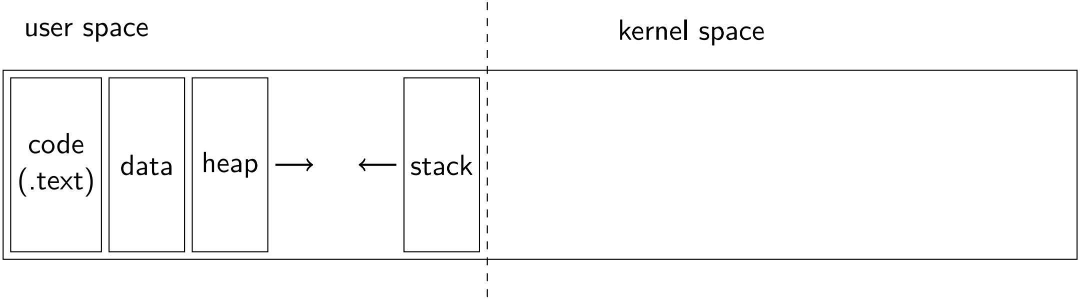


The role of the operating system - provide a virtual environment for a process.

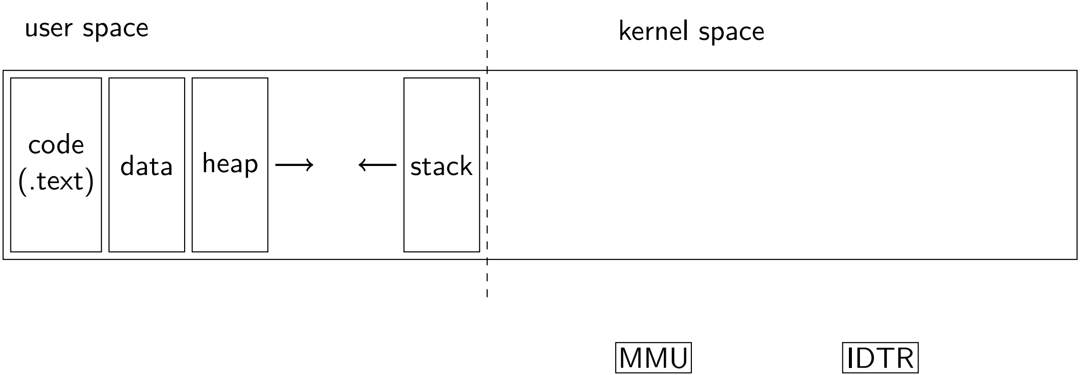
the kernel



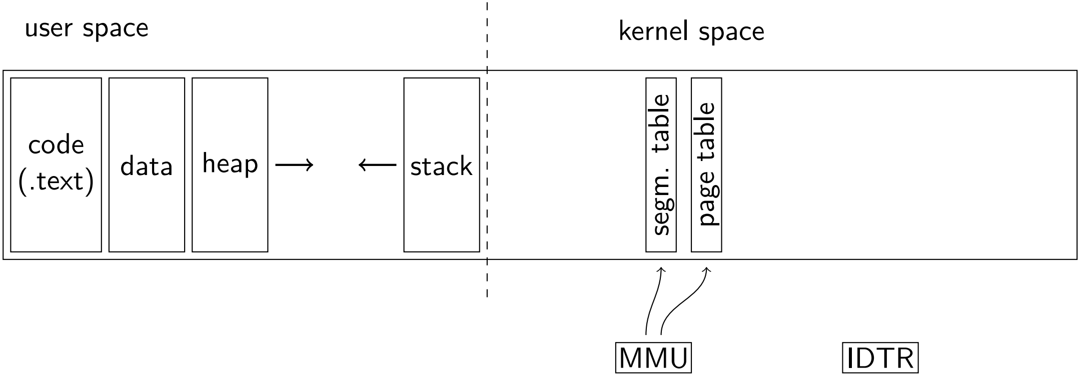
the kernel



the kernel

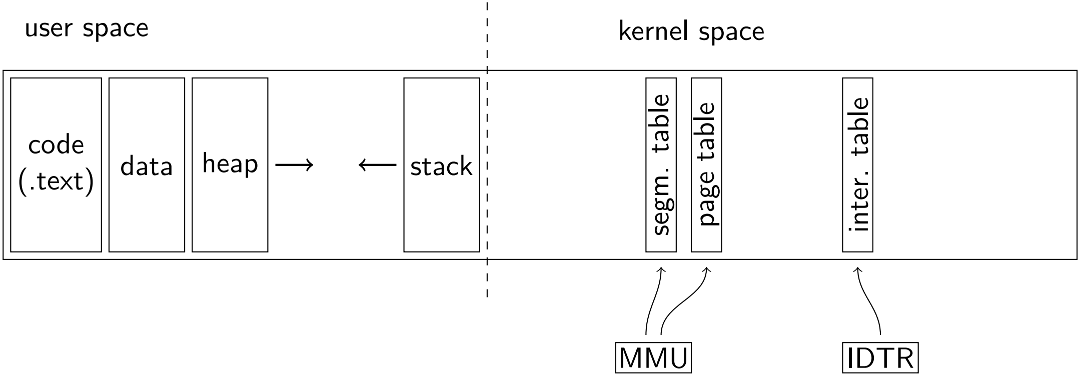


the kernel



user space
kernel space
code
(.text)
data
heap
stack
segm. table
page table
MMU
IDTR

the kernel


user space
kernel space
code
(.text)
data
heap
stack
segm. table
page table
inter. table
MMU
IDTR

indirect execution

Who is in control?

indirect execution

Who is in control?

- control the registers of the MMU and you control the virtual address space

indirect execution

Who is in control?

- control the registers of the MMU and you control the virtual address space
- control the IDTR and you control what will happen when we have an interrupt

Who is in control?

- control the registers of the MMU and you control the virtual address space
- control the IDTR and you control what will happen when we have an interrupt
- instructions to set MMU or IDT registers are privileged instructions

indirect execution

Who is in control?

- control the registers of the MMU and you control the virtual address space
- control the IDTR and you control what will happen when we have an interrupt
- instructions to set MMU or IDT registers are privileged instructions

Limited direct execution:

indirect execution

Who is in control?

- control the registers of the MMU and you control the virtual address space
- control the IDTR and you control what will happen when we have an interrupt
- instructions to set MMU or IDT registers are privileged instructions

Limited direct execution:

- only work with mapped memory in user space,

indirect execution

Who is in control?

- control the registers of the MMU and you control the virtual address space
- control the IDTR and you control what will happen when we have an interrupt
- instructions to set MMU or IDT registers are privileged instructions

Limited direct execution:

- only work with mapped memory in user space,
- only execute non-privileged instructions,

indirect execution

Who is in control?

- control the registers of the MMU and you control the virtual address space
- control the IDTR and you control what will happen when we have an interrupt
- instructions to set MMU or IDT registers are privileged instructions

Limited direct execution:

- only work with mapped memory in user space,
- only execute non-privileged instructions,
- for a limited amount of time.

Interrupts

Synchronous interrupts - exceptions:

- faults:

Interrupts

Synchronous interrupts - exceptions:

- faults:
  - page fault
  - privilege violation
  - divide by zero, ...

Interrupts

Synchronous interrupts - exceptions:

- faults:
  - page fault
  - privilege violation
  - divide by zero, ...
- programmed exceptions:

Interrupts

Synchronous interrupts - exceptions:

- faults:
  - page fault
  - privilege violation
  - divide by zero, ...
- programmed exceptions:
  - system call (`INT 0x80`)
  - debug instructions

Interrupts

Synchronous interrupts - exceptions:

- faults:
  - page fault
  - privilege violation
  - divide by zero, ...
- programmed exceptions:
  - system call (`INT 0x80`)
  - debug instructions

Asynchronous interrupts:

Interrupts

Synchronous interrupts - exceptions:

- faults:
  - page fault
  - privilege violation
  - divide by zero, ...
- programmed exceptions:
  - system call (`INT 0x80`)
  - debug instructions

Asynchronous interrupts:

- timer interrupt

Interrupts

Synchronous interrupts - exceptions:

- faults:
  - page fault
  - privilege violation
  - divide by zero, ...
- programmed exceptions:
  - system call (`INT 0x80`)
  - debug instructions

Asynchronous interrupts:

- timer interrupt
- hardware interrupt: I/O complete, ...

Interrupts

Synchronous interrupts - exceptions:

- faults:
  - page fault
  - privilege violation
  - divide by zero, ...
- programmed exceptions:
  - system call (`INT 0x80`)
  - debug instructions

Asynchronous interrupts:

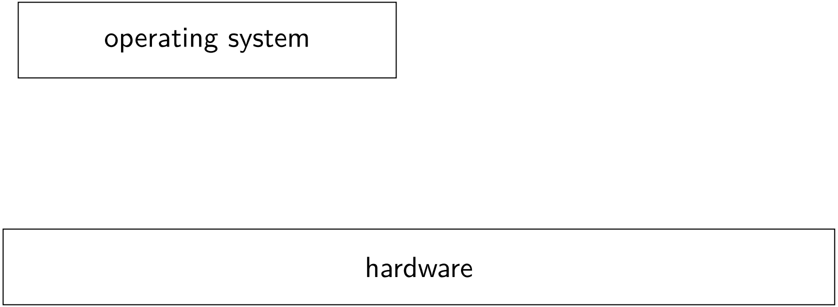
- timer interrupt
- hardware interrupt: I/O complete, ...

The kernel is interrupt driven.

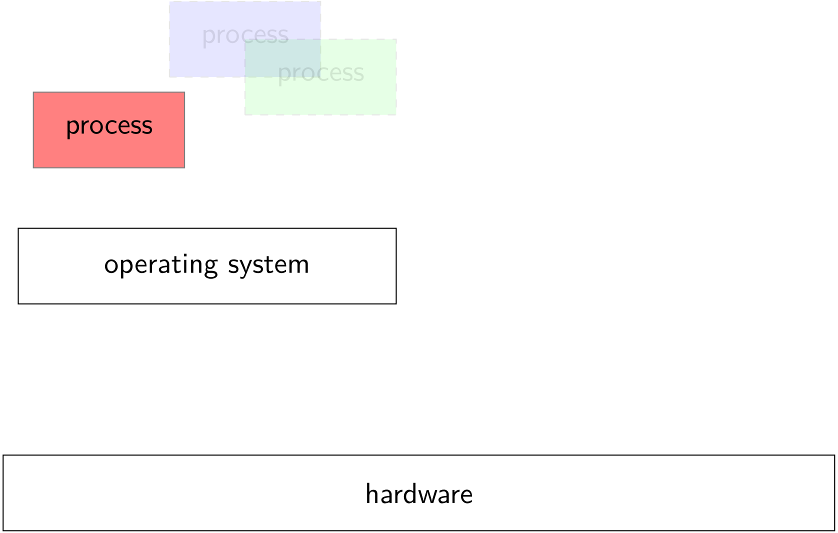
Virtualisation

hardware

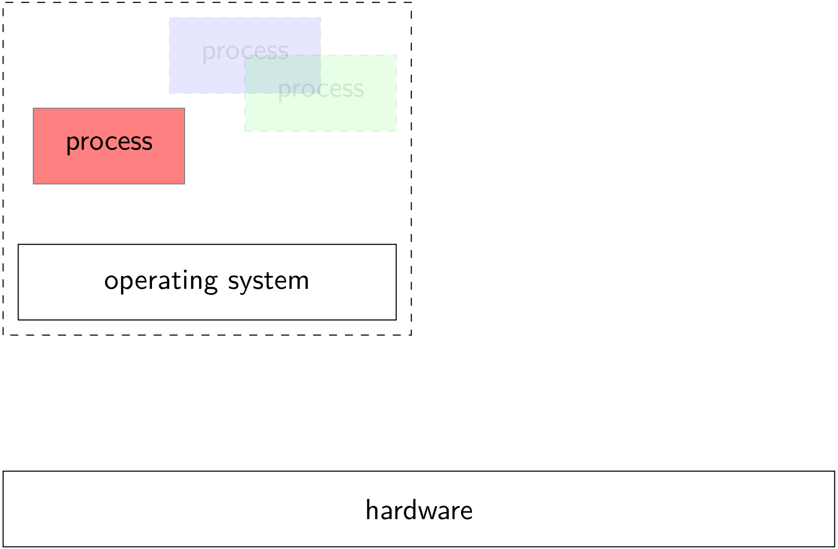
Virtualisation



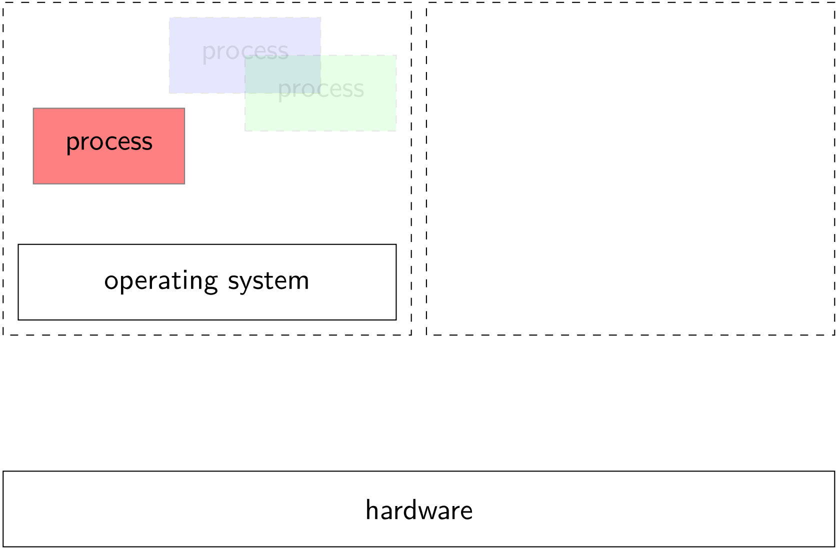
Virtualisation



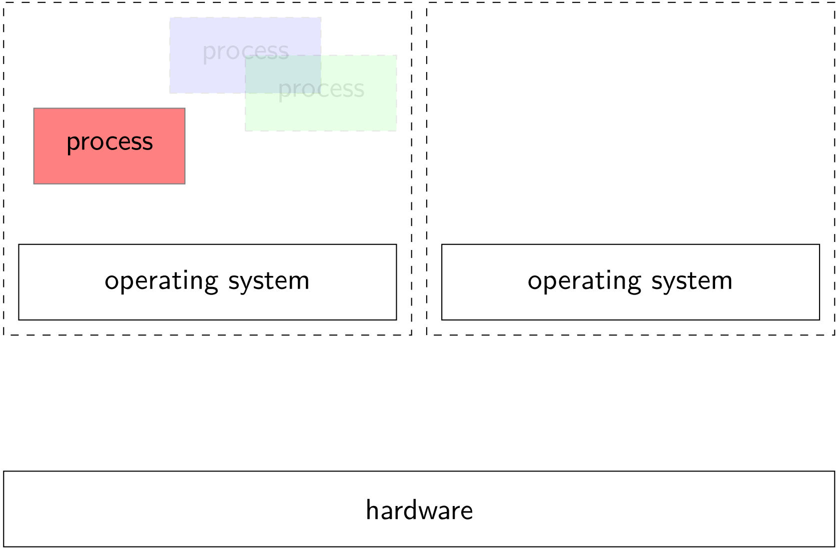
Virtualisation



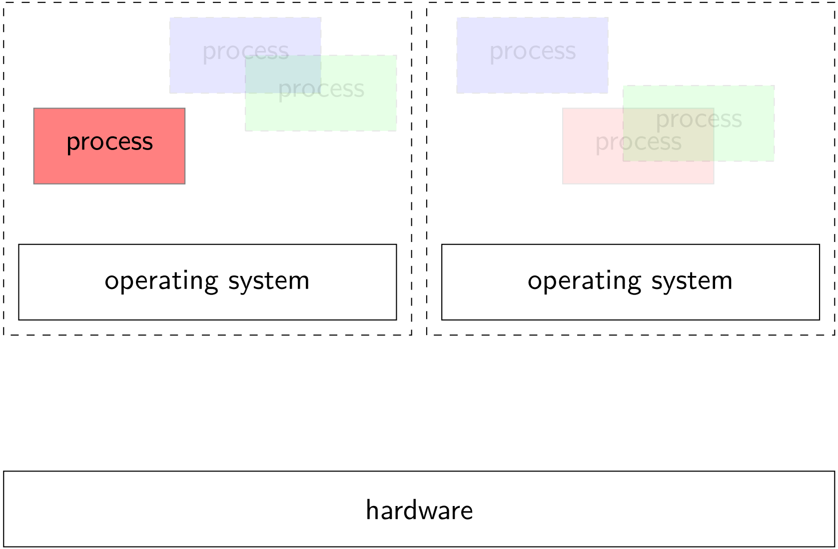
Virtualisation



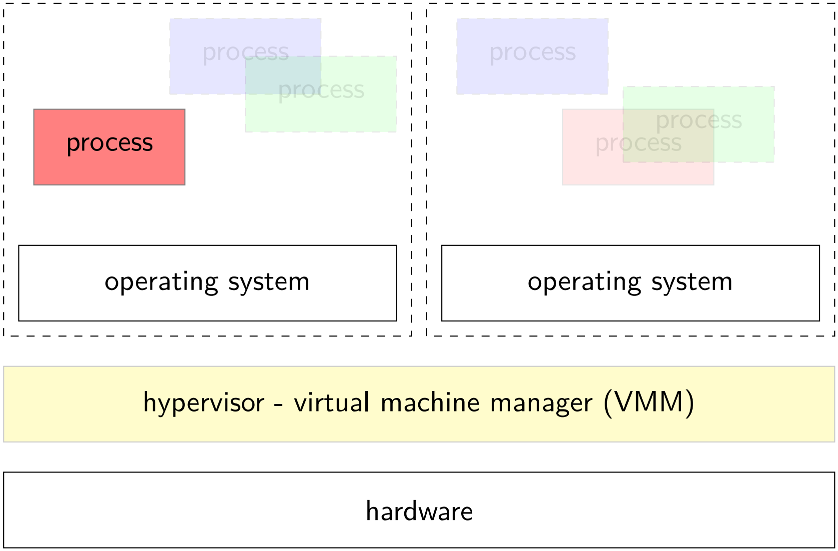
Virtualisation



Virtualisation



Virtualisation



Why?

Why?

Utilisation of hardware.

Why?

Utilisation of hardware.

Also provided by a multi-task operating system, what is new?

Why?

Utilisation of hardware.

Also provided by a multi-task operating system, what is new?

Applications are completely separated from each other.

Why?

Utilisation of hardware.

> Also provided by a multi-task operating system, what is new?

Applications are completely separated from each other.

> What do two processes in an operating system share?

Why?

Utilisation of hardware.

Also provided by a multi-task operating system, what is new?

Applications are completely separated from each other.

What do two processes in an operating system share?

Applications can use different operating systems.

Why?

Utilisation of hardware.

Also provided by a multi-task operating system, what is new?

Applications are completely separated from each other.

What do two processes in an operating system share?

Applications can use different operating systems.

Is this important?

the Hypervisor

Provide virtualisaton of the hardware:

- a virtual cpu, part of the processing power

the Hypervisor

Provide virtualisaton of the hardware:

- a virtual cpu, part of the processing power
- a virtual memory, the illusion of physical memory

the Hypervisor

Provide virtualisaton of the hardware:

- a virtual cpu, part of the processing power
- a virtual memory, the illusion of physical memory

I think we have seen this before.

the solution

Provide *limited direct execution* i.e. allow each guest operating system to execute in *user space* and only perform non-privileged operations.

the solution

Provide *limited direct execution* i.e. allow each guest operating system to execute in *user space* and only perform non-privileged operations.

What is the first thing an operating system wants to do?

the virtual IDT

Hypervisor	Guest Operating system
set up IDT	

the virtual IDT

Hypervisor	Guest Operating system
set up IDT	
pass control to OS	

the virtual IDT

Hypervisor	Guest Operating system
set up IDT	
pass control to OS →	
	initialize OS

the virtual IDT

Hypervisor | Guest Operating system

set up IDT
pass control to OS ⟶

initialize OS
set up IDT

the virtual IDT

Hypervisor		Guest Operating system
set up IDT		
pass control to OS	⟶	
		initialize OS
		set up IDT
handle interrupt	⟵	

the virtual IDT

Hypervisor		Guest Operating system
set up IDT		
pass control to OS	→	
		initialize OS
		set up IDT
handle interrupt	←	
save ref to IDT of OS		

the virtual IDT

Hypervisor		Guest Operating system
set up IDT		
pass control to OS	→	
		initialize OS
		set up IDT
handle interrupt	←	
save ref to IDT of OS		
pass control to OS		

the virtual IDT

Hypervisor		Guest Operating system
set up IDT		
pass control to OS	→	
		initialize OS
		set up IDT
handle interrupt	←	
save ref to IDT of OS		
pass control to OS	→	

the virtual IDT

Hypervisor		Guest Operating system
set up IDT		
pass control to OS	→	
		initialize OS
		set up IDT
handle interrupt	←	
save ref to IDT of OS		
pass control to OS	→	
		continue as if nothing happened

The operating system is running in non-privileged mode.

a system call

Hypervisor | Guest operating system | Application

running

a system call

Hypervisor | Guest operating system | Application

running

system call

a system call

Hypervisor | Guest operating system | Application

running

system call

`INT 0x80`

a system call

Hypervisor | Guest operating system | Application

running

system call

`INT 0x80` ← (arrow from Application to Hypervisor)

a system call

Hypervisor | Guest operating system | Application

running

system call

`INT 0x80` ←

handle interrupt

a system call

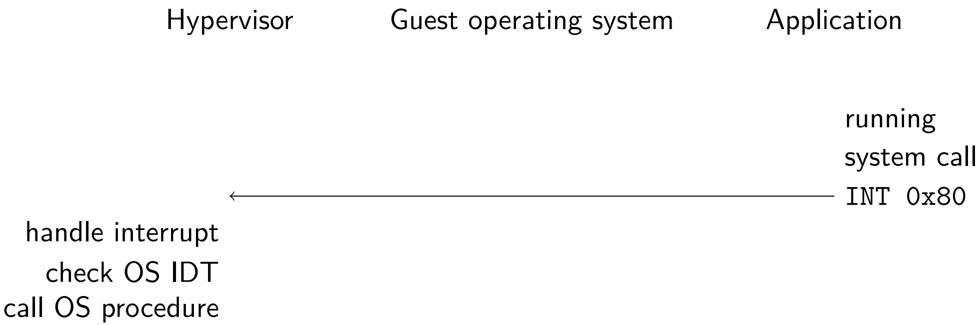
Hypervisor | Guest operating system | Application

running
system call
`INT 0x80`

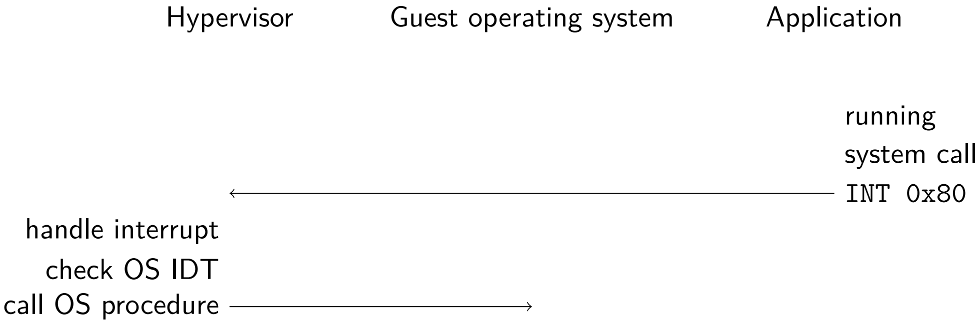
←

handle interrupt
check OS IDT

a system call



a system call



a system call

Hypervisor | Guest operating system | Application

running
system call
`INT 0x80`

handle interrupt
check OS IDT
call OS procedure

handle interrupt

a system call

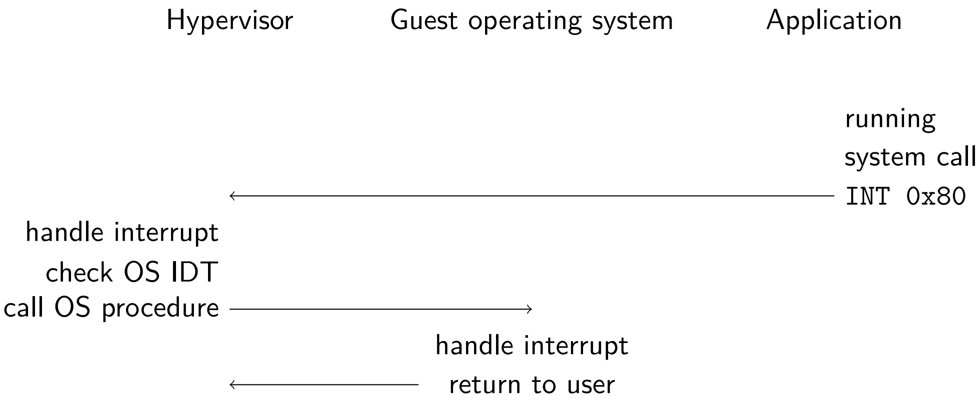
Hypervisor | Guest operating system | Application

running
system call
`INT 0x80` ⟶ (to Hypervisor)

handle interrupt
check OS IDT
call OS procedure ⟶ (to Guest operating system)

handle interrupt
return to user

a system call



a system call

Hypervisor	Guest operating system	Application
		running
		system call
←	←	`INT 0x80`
handle interrupt		
check OS IDT		
call OS procedure →	→	
	handle interrupt	
←	return to user	
handle interrupt		

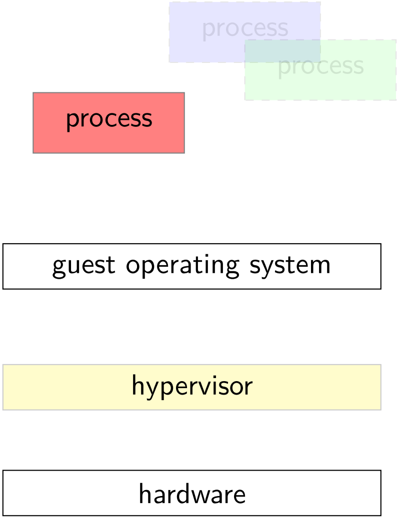
a system call

Hypervisor		Guest operating system		Application
				running
				system call
	←	←	←	`INT 0x80`
handle interrupt				
check OS IDT				
call OS procedure	→			
		handle interrupt		
	←	return to user		
handle interrupt				
return to user	→	→	→	

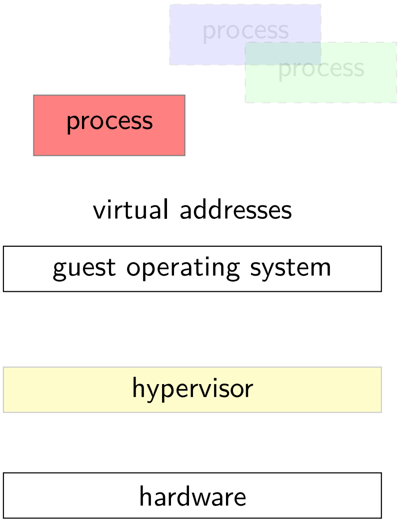
a system call

Hypervisor	Guest operating system	Application
		running
		system call
←	←	`INT 0x80`
handle interrupt		
check OS IDT		
call OS procedure →	→	
	handle interrupt	
←	return to user	
handle interrupt		
return to user →	→	→
		resume execution

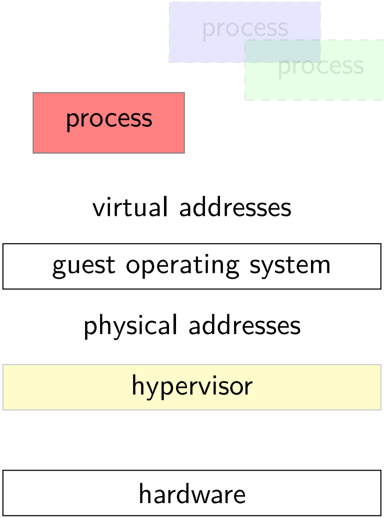
What about virtual memory?



What about virtual memory?

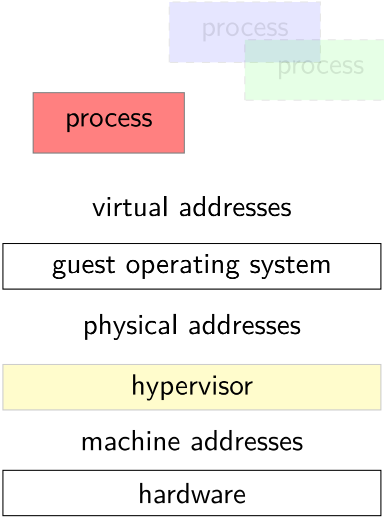


What about virtual memory?



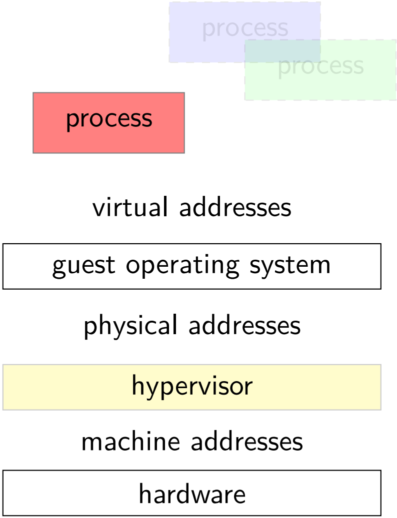
- regular translation tables

What about virtual memory?



- regular translation tables
- second level translation

What about virtual memory?



- regular translation tables
- second level translation

This will be expensive!

regular paging

User process uses virtual addresses that are automatic translated by the hardware (using page table and the MMU) to physical addresses.

regular paging

User process uses virtual addresses that are automatic translated by the hardware (using page table and the MMU) to physical addresses.

A *page fault* invokes the kernel that, if allowed, maps a missing page and return to the user process.

second level paging

Hypervisor | Guest operating system | Application

running

second level paging

Hypervisor | Guest operating system | Application

running
page fault

second level paging

Hypervisor | Guest operating system | Application

running

← page fault

second level paging

Hypervisor | Guest operating system | Application

running

page fault

handle interrupt

second level paging

Hypervisor | Guest operating system | Application

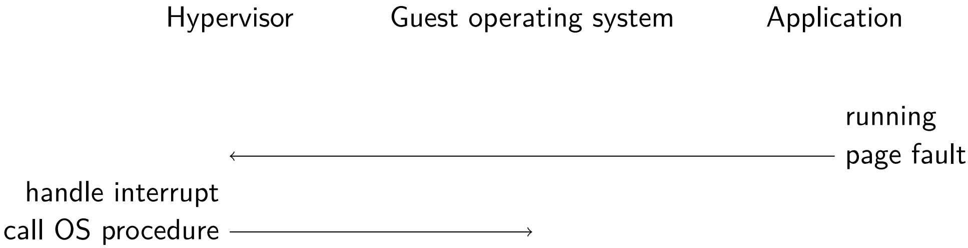
running

page fault (arrow from Application to Hypervisor)

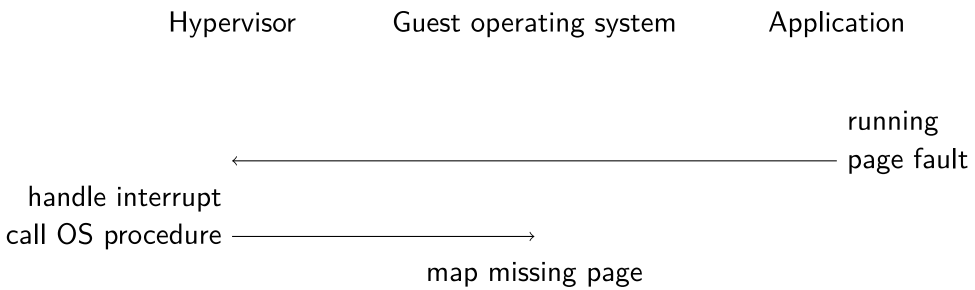
handle interrupt

call OS procedure

second level paging



second level paging



second level paging

Hypervisor | Guest operating system | Application

running

page fault ⟶ (to Hypervisor)

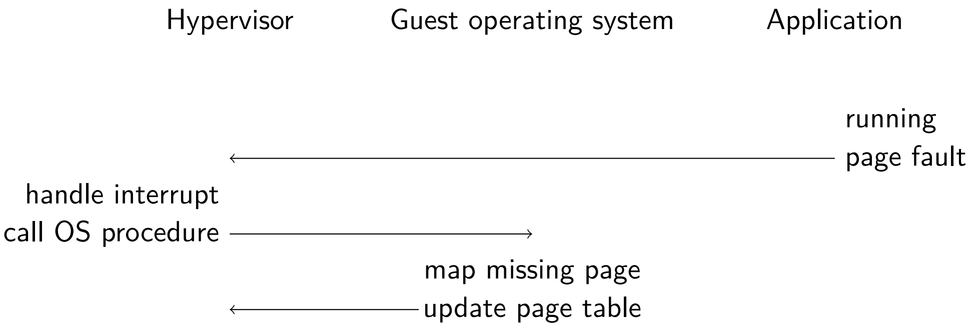
handle interrupt

call OS procedure ⟶ (to Guest operating system)

map missing page

update page table

second level paging



second level paging

Hypervisor — Guest operating system — Application

running

page fault

handle interrupt

call OS procedure

map missing page

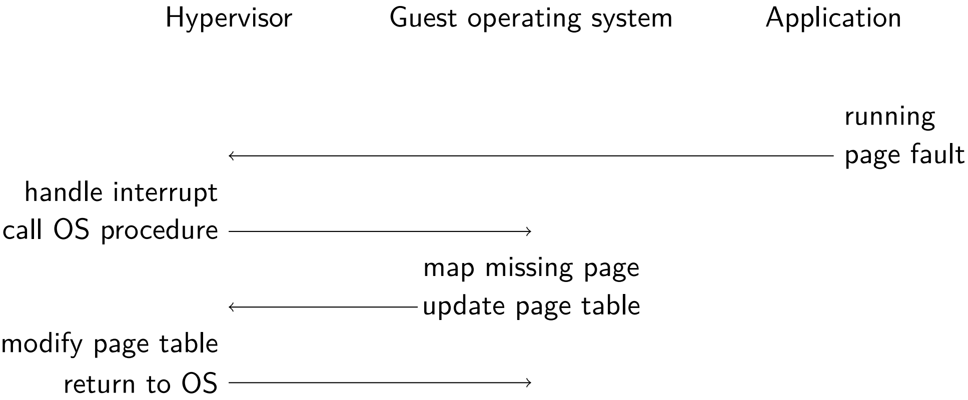
update page table

modify page table

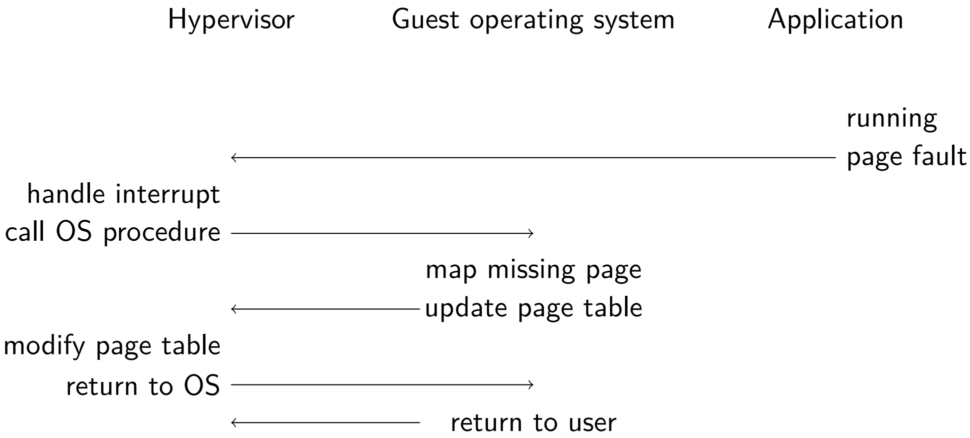
second level paging

Hypervisor	Guest operating system	Application
		running
←	←	page fault
handle interrupt		
call OS procedure →	→	
	map missing page	
←	update page table	
modify page table		
return to OS		

second level paging



second level paging



second level paging

Hypervisor | Guest operating system | Application

running

page fault

handle interrupt

call OS procedure

map missing page

update page table

modify page table

return to OS

return to user

second level paging

Hypervisor	Guest operating system	Application
		running
←	←	page fault
handle interrupt		
call OS procedure →	→	
	map missing page	
←	update page table	
modify page table		
return to OS →	→	
←	return to user	
return to user →	→	→

second level paging

Hypervisor	Guest operating system	Application
		running
←	←	page fault
handle interrupt		
call OS procedure →		
	map missing page	
←	update page table	
modify page table		
return to OS →		
←	return to user	
return to user →	→	
		resume execution

...wait a second

...wait a second

If the guest operating system is executing in user mode - how does it protect itself from the application process that is also running in user mode?

...wait a second

If the guest operating system is executing in user mode - how does it protect itself from the application process that is also running in user mode?

If we allow the guest operating system to run in kernel mode - then the hypervisor can not protect it self.

system call revisited

Hypervisor

Guest operating system

Application

kernel space

user space

in user mode

system call revisited

Hypervisor | Guest operating system | Application

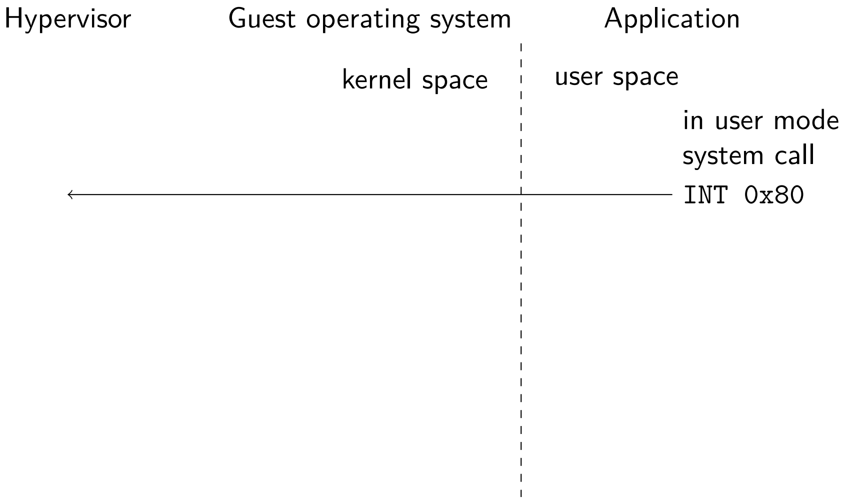
kernel space | user space

in user mode

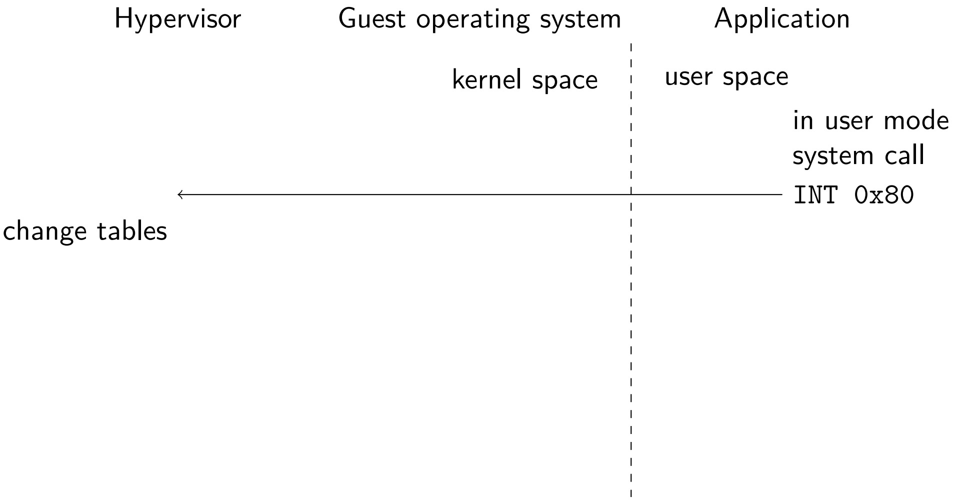
system call

system call revisited

Hypervisor	Guest operating system	Application
	kernel space	user space
		in user mode
		system call
		`INT 0x80`


Hypervisor
Guest operating system
Application
kernel space
user space
in user mode
system call
INT 0x80

system call revisited



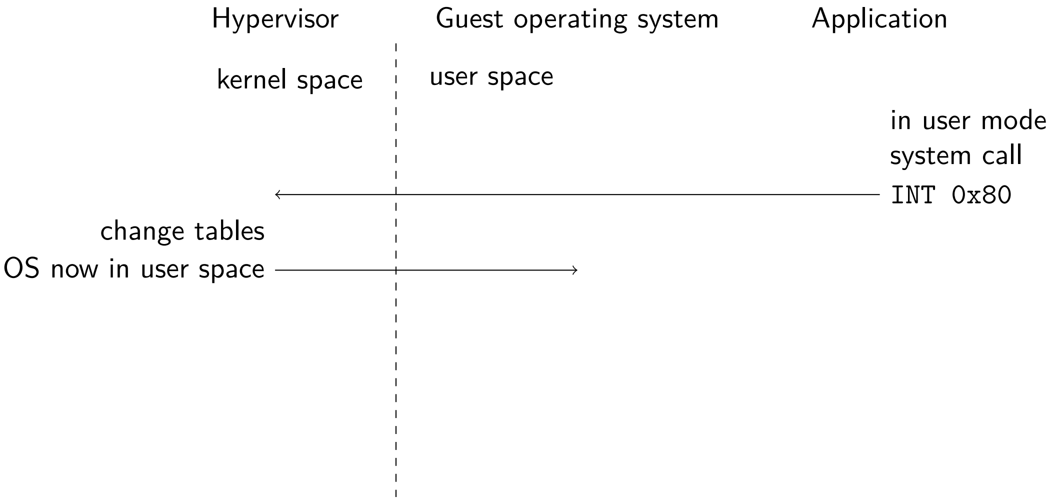
Hypervisor | Guest operating system | Application

kernel space | user space

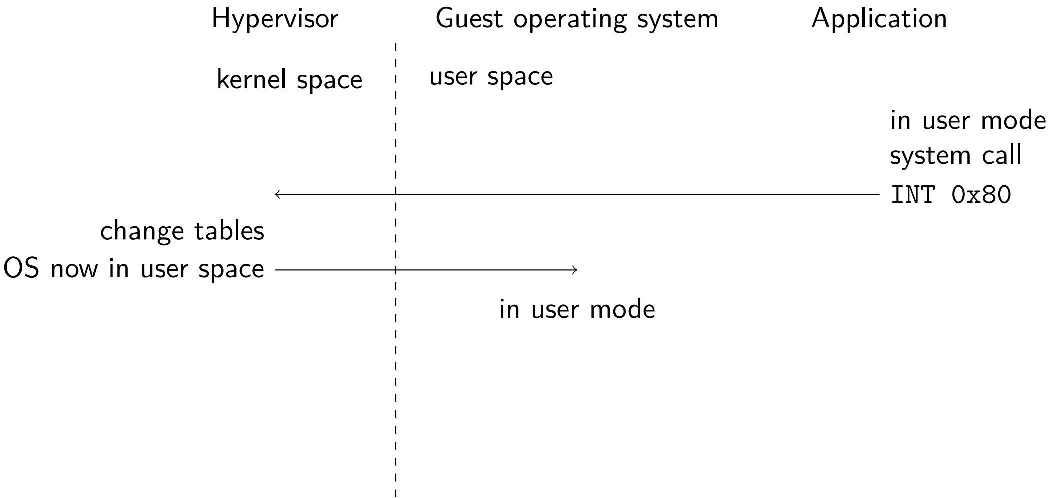
in user mode
system call
`INT 0x80`

change tables
OS now in user space

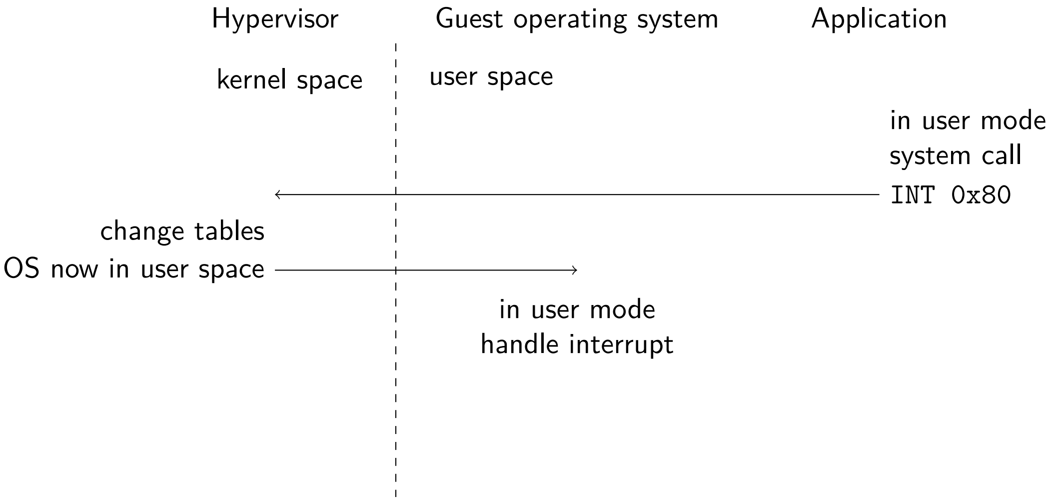
system call revisited



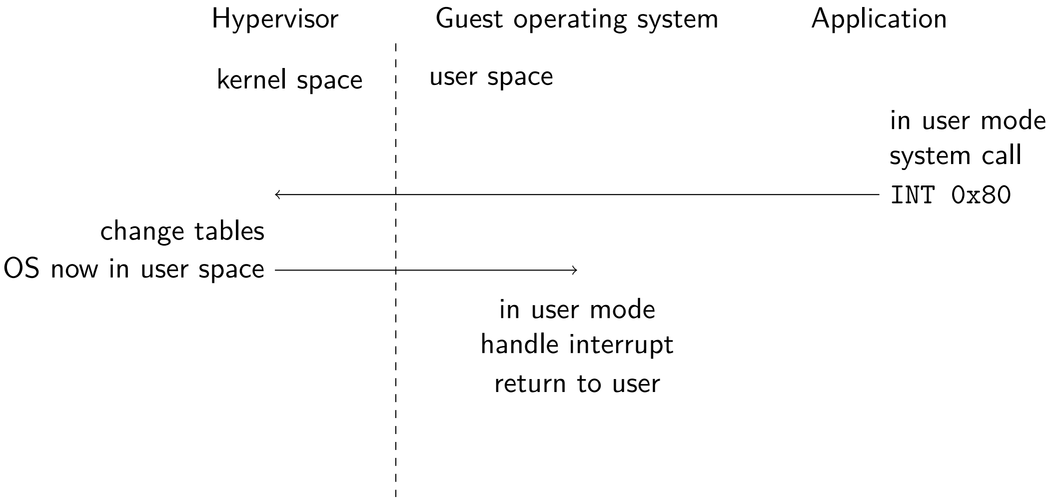
system call revisited



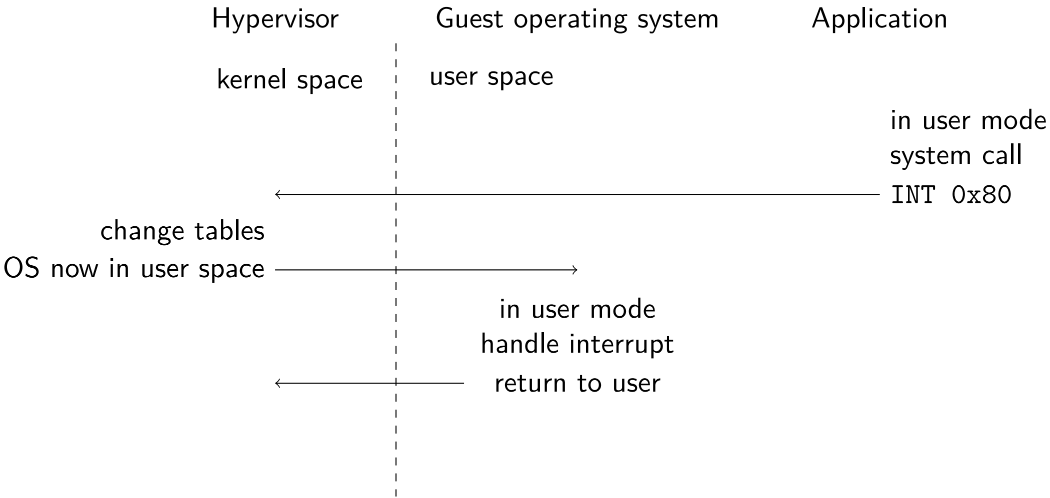
system call revisited



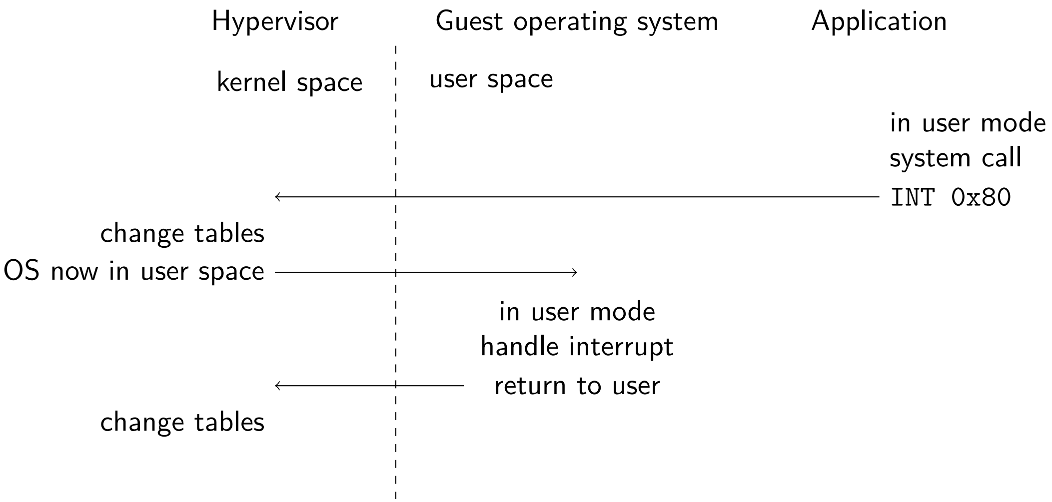
system call revisited



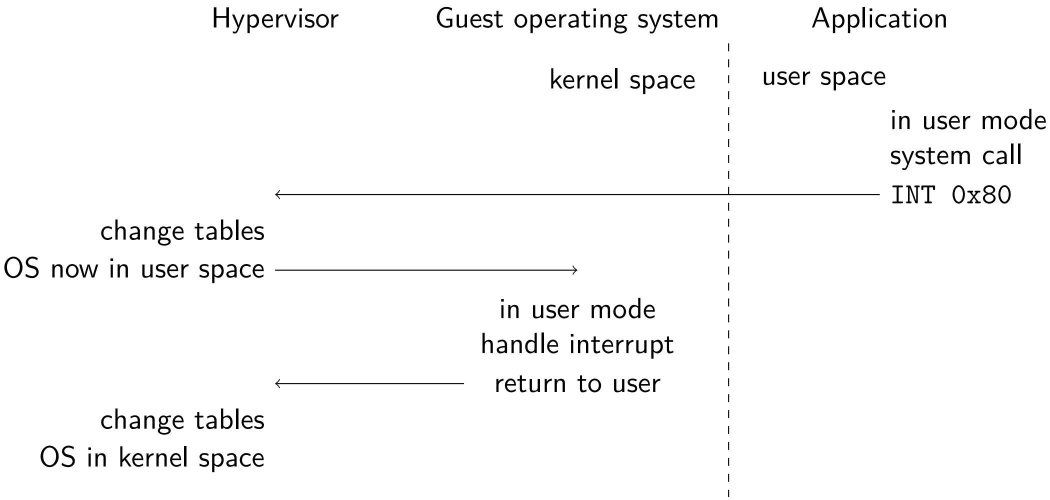
system call revisited



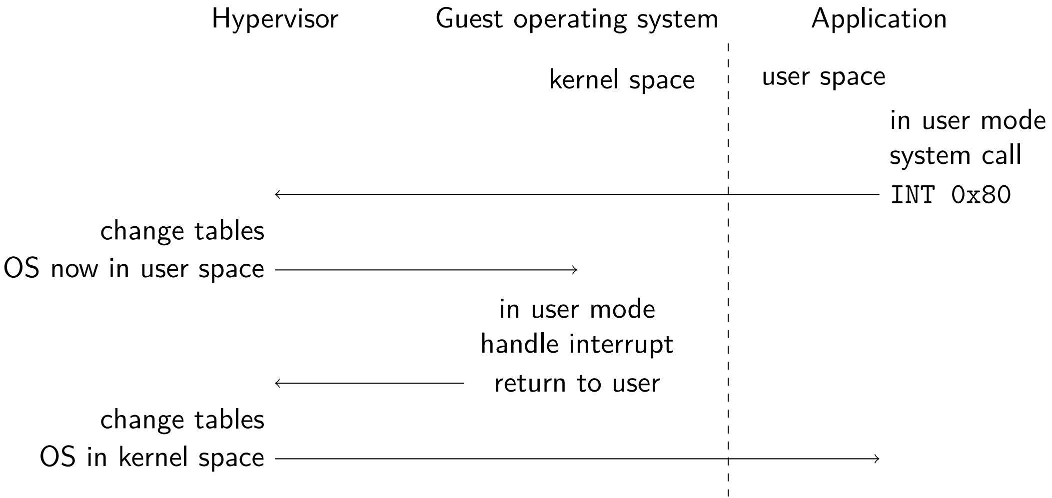
system call revisited



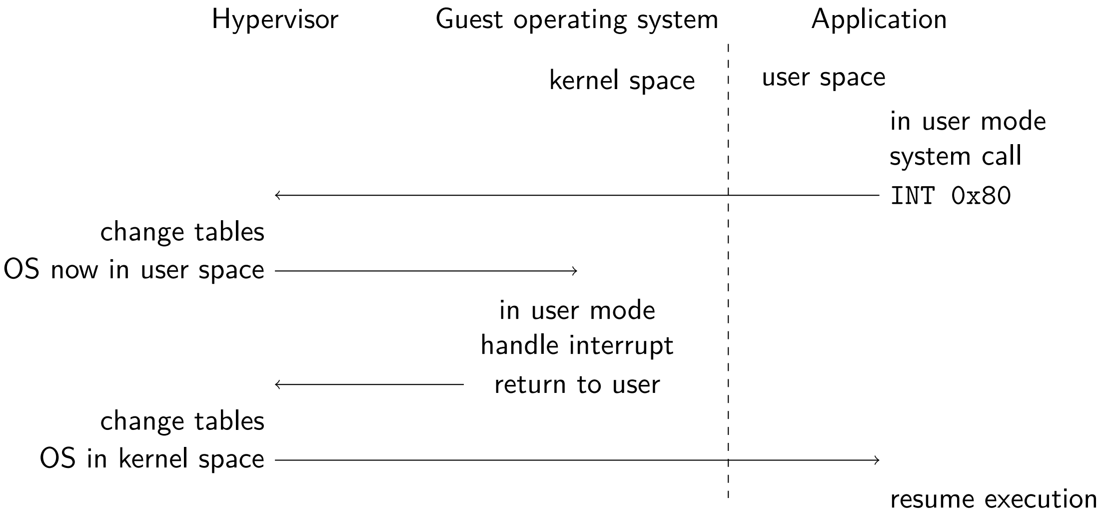
system call revisited



system call revisited



system call revisited



.. thank god for harware

.. thank god for harware

Hardware support:

.. thank god for harware

Hardware support:

- Available in both AMD an Intel x86 processors

.. thank god for harware

Hardware support:

- Available in both AMD an Intel x86 processors
- Allows hypervisors to provide near “bare metal” performance.

.. a different approach

.. a different approach

Para-virtualization: change the operating system that you want to virtualize.

.. a different approach

Para-virtualization: change the operating system that you want to virtualize.

- Change kernel modules in the operating system.

.. a different approach

Para-virtualization: change the operating system that you want to virtualize.

- Change kernel modules in the operating system.

- Recompile source code or patch binary code.

The original goal

The original goal

Utilisation of hardware.

The original goal

Utilisation of hardware.

The original goal

Utilisation of hardware.

Applications are completely separated from each other.

The original goal

Utilisation of hardware.

Applications are completely separated from each other.

The original goal

Utilisation of hardware.

Applications are completely separated from each other.

Applications can use different operating systems.

The original goal

Utilisation of hardware.

Applications are completely separated from each other.

Applications can use different operating systems.

What if we skip this.

containers

An operating system uses several name spaces: memory addresses, file paths, port numbers, device interrupt requests, process id, user id, . . .

containers

An operating system uses several name spaces: memory addresses, file paths, port numbers, device interrupt requests, process id, user id, . . .

Provide a *container*, a separate environment with its own name spaces.

containers

An operating system uses several name spaces: memory addresses, file paths, port numbers, device interrupt requests, process id, user id, . . .

Provide a *container*, a separate environment with its own name spaces.

Processes in different containers are completely separated from each other . . .

containers

An operating system uses several name spaces: memory addresses, file paths, port numbers, device interrupt requests, process id, user id, . . .

Provide a *container*, a separate environment with its own name spaces.

Processes in different containers are completely separated from each other . . . but they use the same kernel.

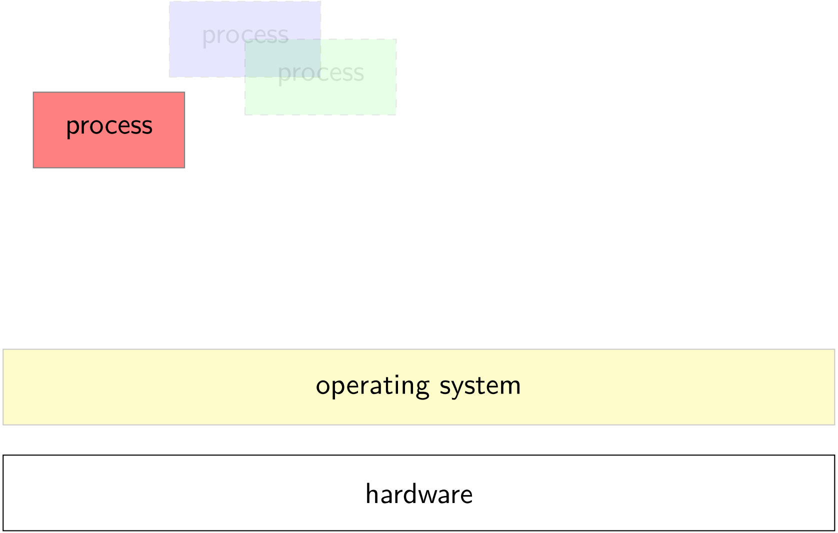
containers

containers

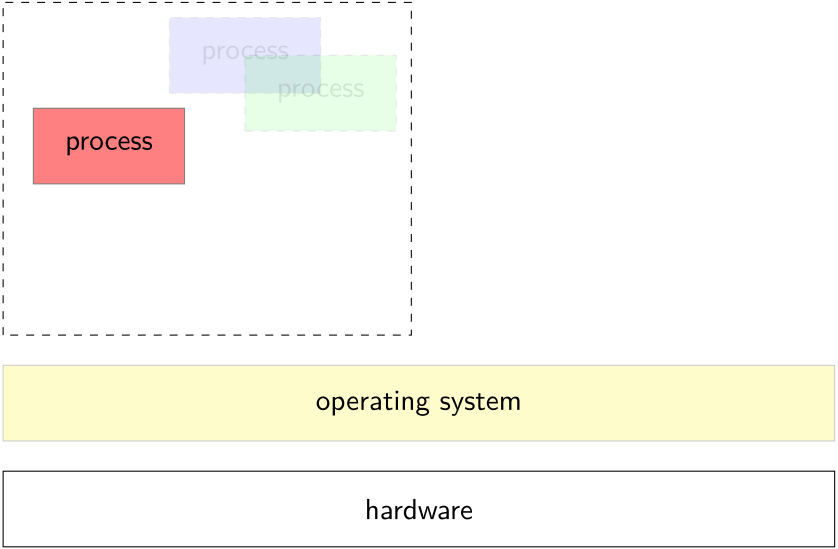
operating system

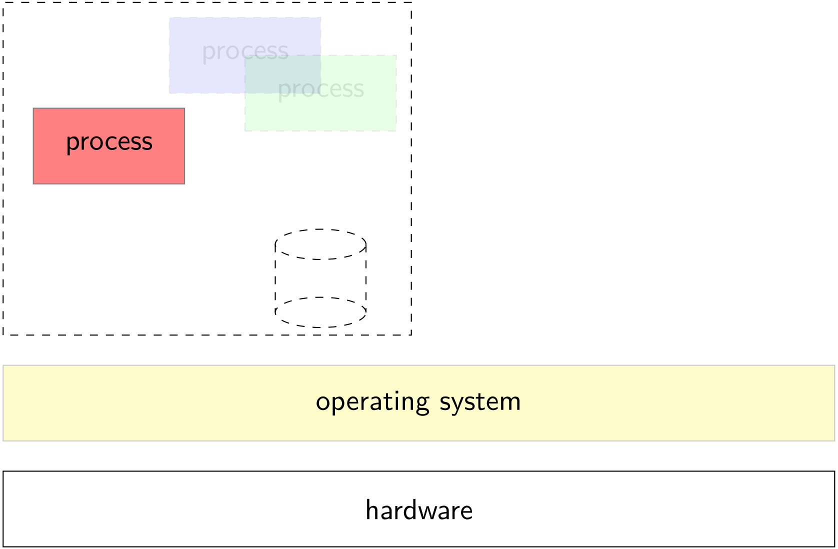
hardware

containers

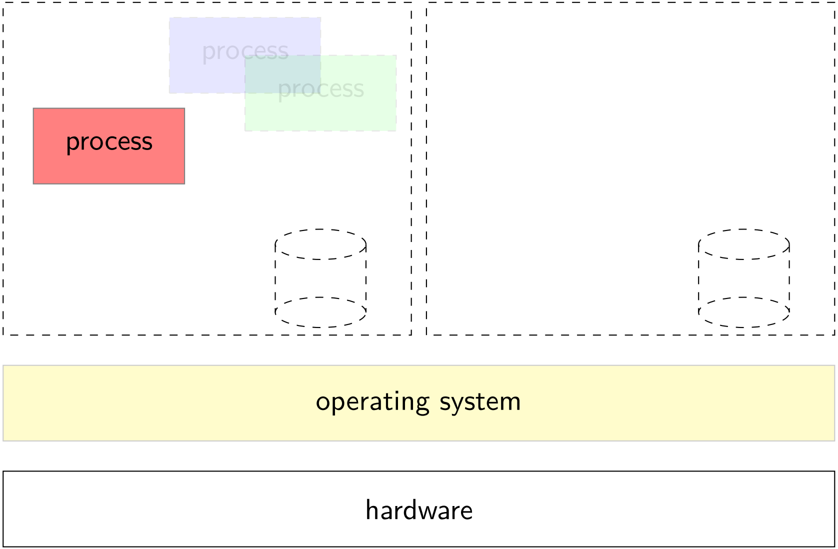


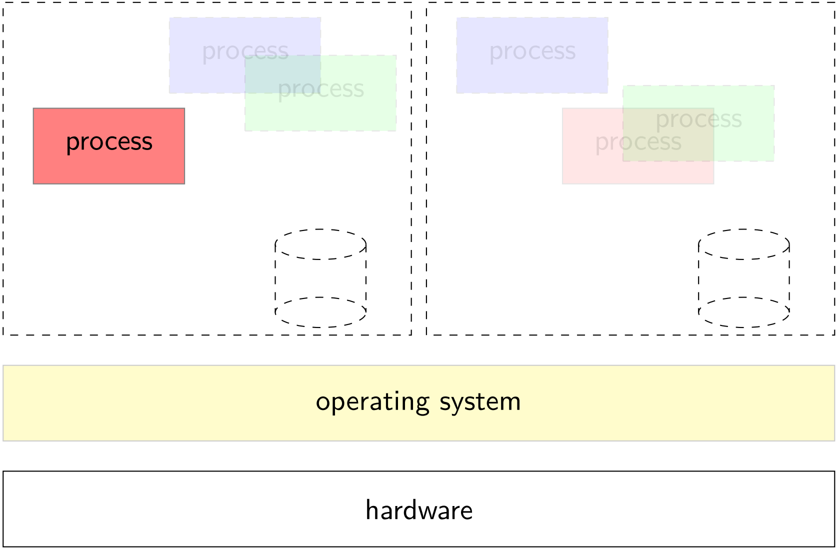
containers




process
process
process
operating system
hardware

containers




process
process
process
process
process
process
operating system
hardware

the original goal

the original goal

Utilisation of hardware.

the original goal

Utilisation of hardware.

the original goal

Utilisation of hardware.

Applications are completely separated from each other.

the original goal

Utilisation of hardware.

Applications are completely separated from each other.

the original goal

Utilisation of hardware.

Applications are completely separated from each other.

Applications can use different operating systems.

Utilisation of hardware.

Applications are completely separated from each other.

Applications can use different operating systems.

Why do they have to run on the same hardware?

emulating hardware

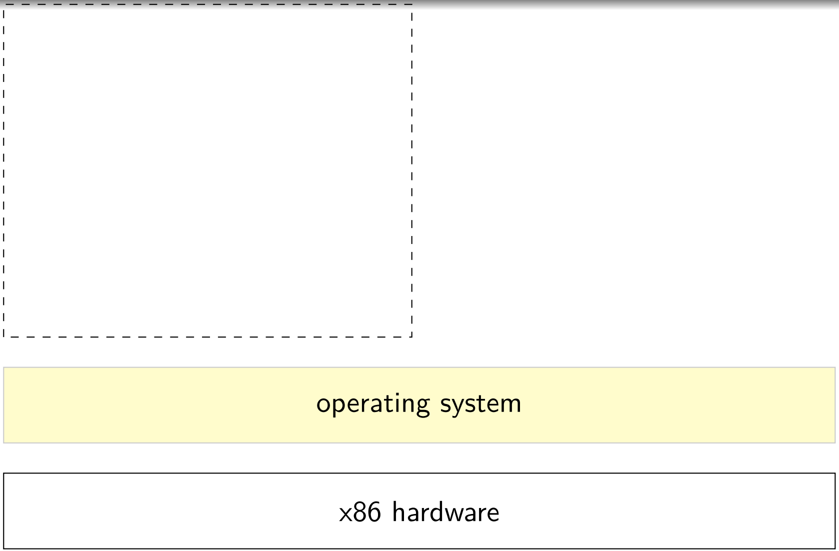
x86 hardware

emulating hardware

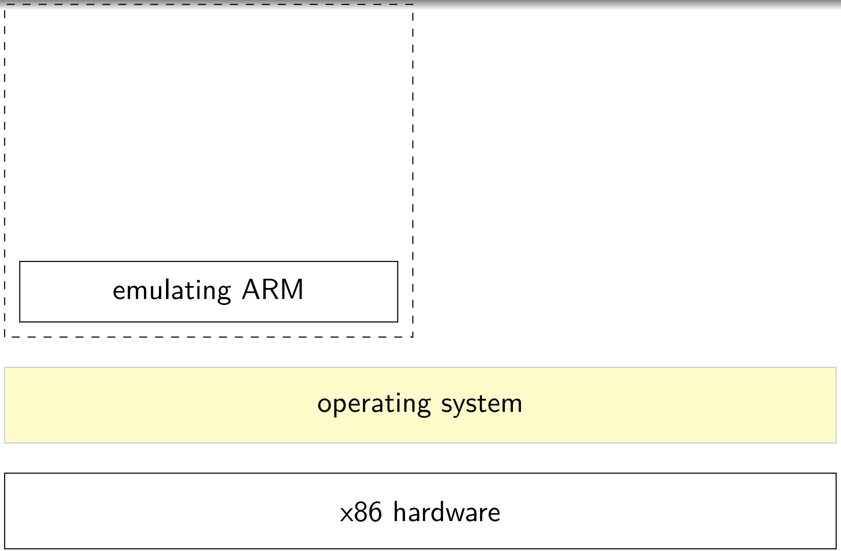
operating system

x86 hardware

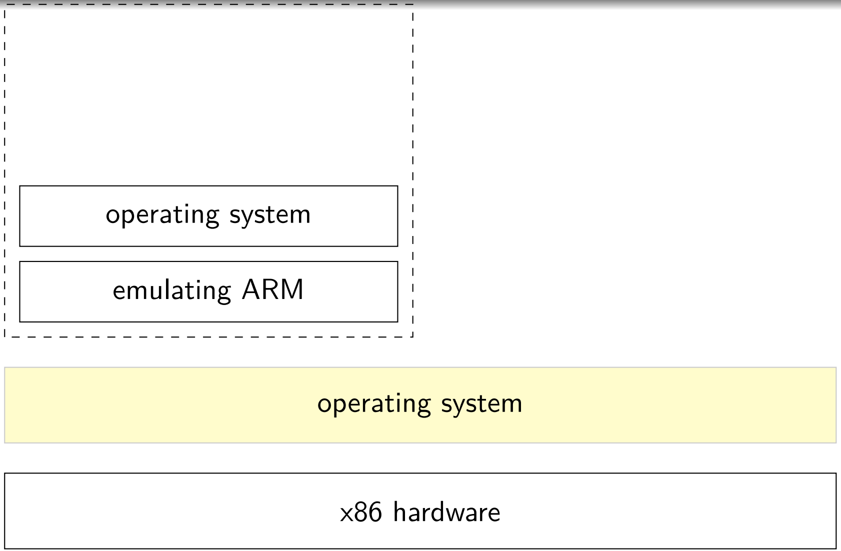
emulating hardware



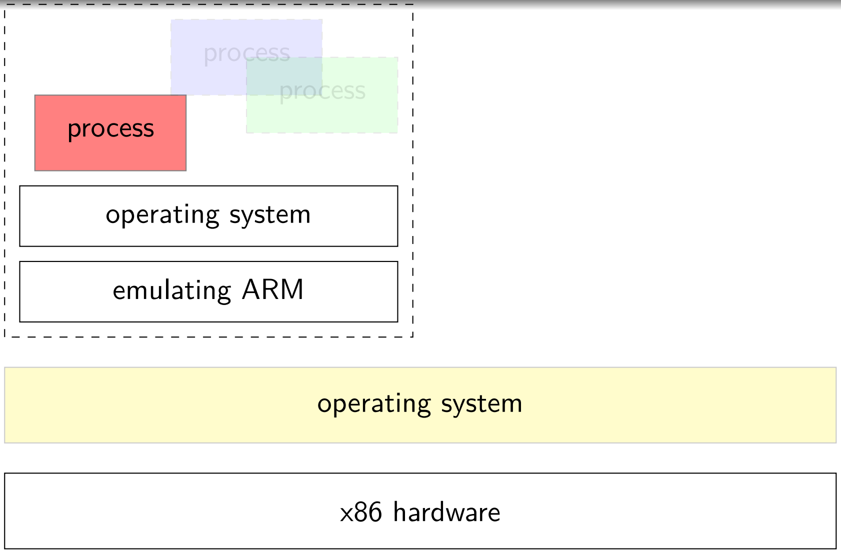
emulating hardware



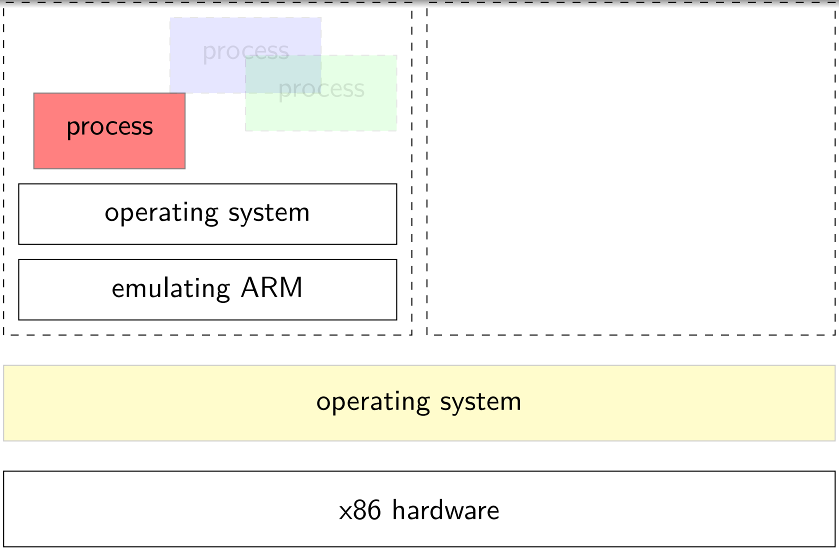
emulating hardware



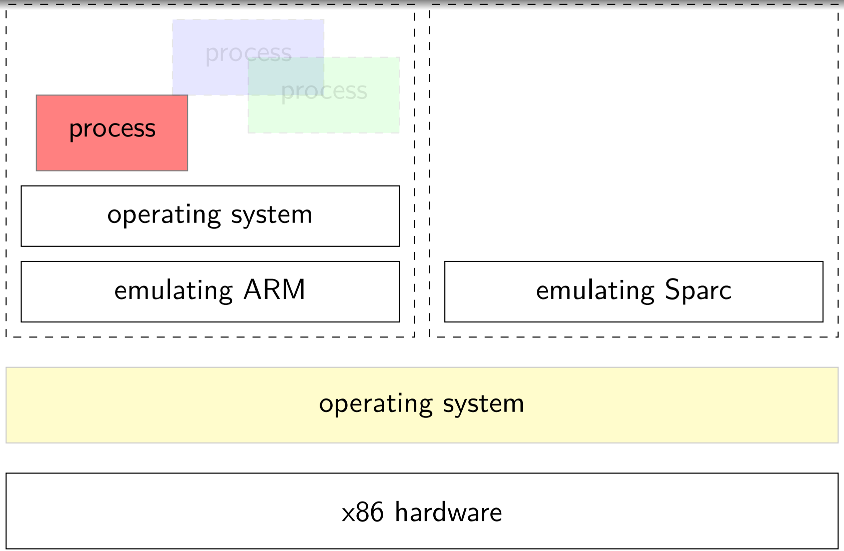
emulating hardware



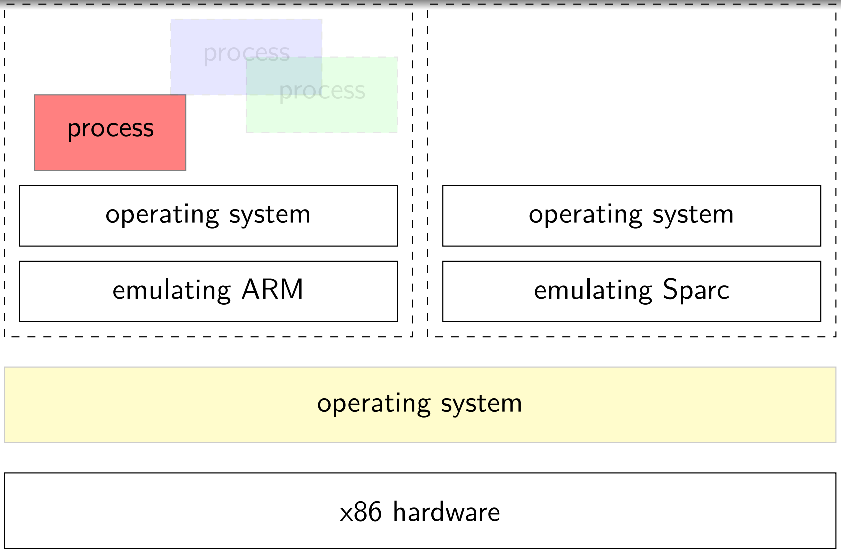
emulating hardware



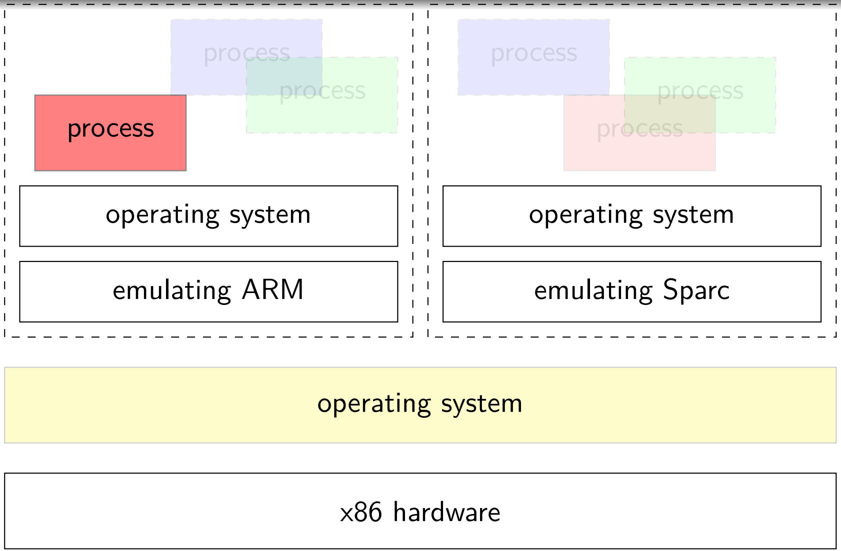
emulating hardware



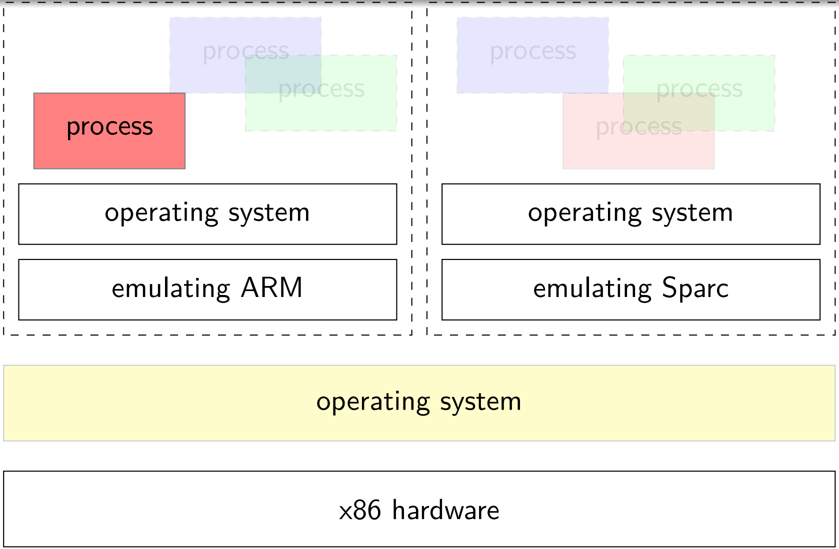
emulating hardware



emulating hardware



emulating hardware



Types of virtual machines

- Emulators
  - Can emulate a different hardware than the host machine (QEMU, Simics).

Types of virtual machines

- Emulators
  - Can emulate a different hardware than the host machine (QEMU, Simics).
- Virtual machines
  - Choose operating system but hardware is set (Xen, KVM, VirtualBox, VMware).

Types of virtual machines

- Emulators
  - Can emulate a different hardware than the host machine (QEMU, Simics).
- Virtual machines
  - Choose operating system but hardware is set (Xen, KVM, VirtualBox, VMware).
- Containers
  - Separated name spaces in the same operating system (Dockers, Linux Containers).

Types of virtual machines

- Emulators
  - Can emulate a different hardware than the host machine (QEMU, Simics).
- Virtual machines
  - Choose operating system but hardware is set (Xen, KVM, VirtualBox, VMware).
- Containers
  - Separated name spaces in the same operating system (Dockers, Linux Containers).
- Runtime systems
  - Dedicated to a language (JVM, Erlang).

... but I never installed a Hypervisor?

… but I never installed a Hypervisor?

VirtualBox etc also installs a kernel module that turns your regular operating system into a hypervisor.

Summary

- Multiple operating systems running on the same machine.

Summary

- Multiple operating systems running on the same machine.
- Each operating system provided a virtual hardware.

Summary

- Multiple operating systems running on the same machine.
- Each operating system provided a virtual hardware.
- With hardware support, near bare metal execution speed can be obtained.

Summary

- Multiple operating systems running on the same machine.
- Each operating system provided a virtual hardware.
- With hardware support, near bare metal execution speed can be obtained.
- Other types: emulators, containers, runtime environments.

Summary

- Multiple operating systems running on the same machine.
- Each operating system provided a virtual hardware.
- With hardware support, near bare metal execution speed can be obtained.
- Other types: emulators, containers, runtime environments.