Virtualisation

Johan Montelius

KTH

2018
The role of the operating system - provide a virtual environment for a process.
the kernel

user space

code (.text)  data  heap  →  stack

kernel space
the kernel code (.text) data heap stack

user space

kernel space

MMU IDTR
the kernel

user space

code (.text)  data  heap  →  stack

kernel space

segm. table  page table

MMU  IDTR
the kernel

user space

- code (.text)
- data
- heap
- stack

kernel space

- segm. table
- page table
- inter. table

MMU

IDTR
indirect execution

Who is in control?
Who is in control?

- control the registers of the MMU and you control the virtual address space
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
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
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
间接执行

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,
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.
Synchronous interrupts - exceptions:
- faults:
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
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.
Interrupts

Synchronous interrupts - exceptions:

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

- programmed exceptions:
  - system call (INT 0x80)
  - debug instructions

Asynchronous interrupts:

- timer interrupt
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, ...
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

- operating system
- hardware
Virtualisation

process

operating system

hardware
Virtualisation
Virtualisation

- process
- operating system
- hardware
Virtualisation

- process
- operating system
- hypervisor - virtual machine manager (VMM)
- hardware
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?
Why?

Utilisation of hardware.
Utilisation of hardware.

Also provided by a multi-task operating system, what is new?
Utilisation of hardware.

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

Applications are completely separated from each other.
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?
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?
Provide virtualisation of the hardware:

- a virtual CPU, part of the processing power
the Hypervisor

Provide virtualisation of the hardware:

- a virtual cpu, part of the processing power
- a virtual memory, the illusion of physical memory
Provide virtualisation 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.*
Provide *limited direct execution* i.e. allow each guest operating system to execute in *user space* and only perform non-privileged operations.
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
Hypervisor

set up IDT
pass control to OS

Guest Operating system
The virtual IDT

Hypervisor

Guest Operating system

set up IDT
pass control to OS
initialize OS
The operating system is running in non-privileged mode.

Initial steps:
- Hypervisor
  - set up IDT
  - pass control to OS
- Guest Operating system
  - initialize OS
  - set up IDT
The virtual IDT

Hypervisor

- set up IDT
- pass control to OS
- handle interrupt

Guest Operating system

- initialize OS
- set up IDT
The virtual IDT

Hypervisor

- set up IDT
- pass control to OS

Guest Operating system

- initialize OS
- set up IDT

handle interrupt

save ref to IDT of OS
The virtual IDT

Hypervisor

set up IDT
pass control to OS
handle interrupt
save ref to IDT of OS
pass control to OS

Guest Operating system

initialize OS
set up IDT
the virtual IDT

Hypervisor

- set up IDT
- pass control to OS
- handle interrupt
- save ref to IDT of OS
- pass control to OS

Guest Operating system

- initialize OS
- set up IDT
The operating system is running in non-privileged mode.
<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>Guest operating system</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>running</td>
</tr>
<tr>
<td>Hypervisor</td>
<td>Guest operating system</td>
<td>Application</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>running system call</td>
</tr>
<tr>
<td>Hypervisor</td>
<td>Guest operating system</td>
<td>Application</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>running system call</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INT 0x80</td>
</tr>
</tbody>
</table>
Hypervisor  Guest operating system  Application

running system call

INT 0x80
a system call

Hypervisor  Guest operating system  Application

running system call

INT 0x80

handle interrupt
A system call involves the following steps:

1. The application runs a system call.
2. The system call is handled by the guest operating system.
3. The guest operating system checks the OS IDT.
4. The hypervisor handles the interrupt.

The system call is indicated by INT 0x80.
a system call

Hypervisor                  Guest operating system                  Application

Running system call

INT 0x80

handle interrupt
check OS IDT
call OS procedure
a system call

Hypervisor  Guest operating system  Application

handle interrupt  check OS IDT  call OS procedure

running system call

INT 0x80
a system call

Hypervisor  Guest operating system  Application

handle interrupt  
check OS IDT  
call OS procedure  
run system call
INT 0x80  
handle interrupt
A system call involves several steps:

1. The application runs a system call, typically through an interrupt (INT 0x80).
2. The hypervisor handles the interrupt.
3. The hypervisor checks the operating system's IDT (Interrupt Descriptor Table) to determine which procedure to call.
4. The hypervisor calls the appropriate OS procedure.
5. The OS procedure is executed.
6. The OS procedure handles the interrupt.
7. The hypervisor returns to the user application.
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

11 / 26
a system call

Hypervisor   Guest operating system   Application

running system call

INT 0x80

handle interrupt
check OS IDT
call OS procedure

handle interrupt

handle interrupt

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
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

resume execution
What about virtual memory?

- process
- guest operating system
- hypervisor
- hardware

This will be expensive!
What about virtual memory?

- virtual addresses
- guest operating system
- hypervisor
- hardware
What about virtual memory?

- regular translation tables

- virtual addresses
  - guest operating system
  - physical addresses
  - hypervisor

- hardware
What about virtual memory?

- regular translation tables
- second level translation

virtual addresses

guest operating system

physical addresses

hypervisor

machine addresses

hardware
What about virtual memory?

- virtual addresses
- guest operating system
- physical addresses
- hypervisor
- machine addresses
- hardware

- regular translation tables
- second level translation

This will be expensive!
User process uses virtual addresses that are automatic translated by the hardware (using page table and the MMU) to physical addresses.
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.
<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>Guest operating system</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>running</td>
</tr>
</tbody>
</table>
## second level paging

<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>Guest operating system</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>running</td>
<td>running page fault</td>
</tr>
</tbody>
</table>
second level paging

Hypervisor    Guest operating system    Application

running

page fault
Hypervisor  Guest operating system  Application

running page fault

handle interrupt
second level paging

Hypervisor  Guest operating system  Application

running page fault

download interrupt
call OS procedure
second level paging

Hypervisor  Guest operating system  Application

running page fault

handle interrupt
call OS procedure
second level paging

Hypervisor          Guest operating system          Application

handle interrupt    running page fault

call OS procedure   map missing page
second level paging

Hypervisor | Guest operating system | Application

handle interrupt | call OS procedure | running page fault

map missing page
update page table
second level paging

Hypervisor       Guest operating system       Application

running
page fault

handle interrupt

call OS procedure

map missing page

update page table
second level paging

Hypervisor → Guest operating system ← Application

- handle interrupt
- call OS procedure
- map missing page
- update page table
- modify page table

running page fault
Hypervisor | Guest operating system | Application

- handle interrupt
- call OS procedure
- map missing page
- update page table
- modify page table
- return to OS

running page fault
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

Hypervisor  Guest operating system  Application

handle interrupt  running page fault

call OS procedure  map missing page

modify page table  update 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
second level paging

Hypervisor → Guest operating system → Application

running page fault

handle interrupt

call OS procedure

map missing page

modify page table

update page table

return to OS

return to user

return to user
second level paging

<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>Guest operating system</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle interrupt</td>
<td>map missing page</td>
<td>running page fault</td>
</tr>
<tr>
<td>call OS procedure</td>
<td>update page table</td>
<td></td>
</tr>
<tr>
<td>modify page table</td>
<td>return to OS</td>
<td></td>
</tr>
<tr>
<td>return to user</td>
<td>return to user</td>
<td></td>
</tr>
<tr>
<td>return to user</td>
<td>resume execution</td>
<td></td>
</tr>
</tbody>
</table>
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 cannot protect itself.
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 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 cannot protect itself.
## System Call Revisited

<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>Guest Operating System</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>kernel space</td>
<td>user space</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in user mode</td>
</tr>
<tr>
<td>Hypervisor</td>
<td>Guest operating system</td>
<td>Application</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>kernel space</td>
<td>user space</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in user mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system call</td>
</tr>
<tr>
<td>Hypervisor</td>
<td>Guest operating system</td>
<td>Application</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>kernel space</td>
<td>user space</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in user mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system call</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INT 0x80</td>
</tr>
</tbody>
</table>
system call revisited

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

change tables

in user mode

system call

INT 0x80
system call revisited

Hypervisor
kernel space
change tables
OS now in user space

Guest operating system
user space

Application
in user mode
system call
INT 0x80
system call revisited

Hypervisor

kernel space

Guest operating system

user space

Application

in user mode

system call

INT 0x80

change tables

OS now in user space
system call revisited

Hypervisor
kernel space

Guest operating system
user space

Application

in user mode
system call
INT 0x80

change tables
OS now in user space

in user mode
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

in user mode

handle interrupt
system call revisited

- Hypervisor
  - kernel space
  - change tables
  - OS now in user space

- Guest operating system
  - user space

- Application
  - in user mode
  - system call
  - INT 0x80

- in user mode
- handle interrupt
- return to user
system call revisited

Hypervisor
kernel space

Guest operating system
user space

Application

in user mode
system call
INT 0x80

→

change tables
OS now in user space

in user mode
handle interrupt

→

change tables
return to user
system call revisited

<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>Guest operating system</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel space</td>
<td>user space</td>
<td>in user mode system call</td>
</tr>
<tr>
<td>change tables</td>
<td></td>
<td>INT 0x80</td>
</tr>
</tbody>
</table>

OS now in user space

in user mode handle interrupt

change tables

OS in kernel space

return to user
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

in user mode
handle interrupt

change tables
OS in kernel space

return to user
System call revisited

Hypervisor

Guest operating system

Application

Kernel space

User space

in user mode

system call

INT 0x80

in user mode

handle interrupt

return to user

resume execution

change tables

OS now in user space

change tables

OS in kernel space

handle interrupt
.. thank god for hardware

Hardware support:
- Available in both AMD and Intel x86 processors
- Allows hypervisors to provide near "bare metal" performance.
Hardware support:
Hardware support:

- Available in both AMD and Intel x86 processors
Hardware support:

- Available in both AMD and Intel x86 processors
- Allows hypervisors to provide near “bare metal” performance.
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.
Para-virtualization: change the operating system that you want to virtualize.
Para-virtualization: change the operating system that you want to virtualize.

- Change kernel modules in the operating system.
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.
An operating system uses several namespaces: memory addresses, file paths, port numbers, device interrupt requests, process id, user id, ...

Provide a container, a separate environment with its own namespaces. Processes in different containers are completely separated from each other ...

...but they use the same kernel.
An operating system uses several name spaces: memory addresses, file paths, port numbers, device interrupt requests, process id, user id, ...
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.
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 ...
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

operating system

hardware
containers

process

operating system

hardware
containers

operating system

hardware
containers

process

process

process

process

process

operating system

hardware
the original goal
Utilisation of hardware.
Utilisation of hardware.
Utilisation of hardware.

Applications are completely separated from each other.
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

Hardware emulators can be surprisingly efficient.

operating system

x86 hardware
Hardware emulators can be surprisingly efficient.
Hardware emulators can be surprisingly efficient.

emulating hardware

- emulating ARM
- operating system
- x86 hardware
Hardware emulators can be surprisingly efficient.
emulating hardware

Hardware emulators can be surprisingly efficient.
Hardware emulators can be surprisingly efficient.
Hardware emulators can be surprisingly efficient.
Hardware emulators can be surprisingly efficient.
Hardware emulators can be surprisingly efficient.
Hardware emulators can be surprisingly efficient.
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?
VirtualBox etc also installs a kernel module that turns your regular operating system into a hypervisor.
Multiple operating systems running on the same machine.
Multiple operating systems running on the same machine.
Each operating system provided a virtual hardware.
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.
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.
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.