Virtualisation

Johan Montelius

KTH

2017
The role of the operating system - provide a virtual environment for a process.
the kernel code (.text) data heap stack kernel space

user space
The kernel

code (.text)  data  heap  stack

user space

kernel space

MMU  IDTR
The kernel code (.text) is in the user space. It is followed by the data, heap, and stack. In the kernel space, there is the segment table and the page table, managed by the MMU (Memory Management Unit) and IDTR (Interrupt Descriptor Table Register).
the kernel

user space

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

kernel space

- segm. table
- page table
- inter. table

architecture components:
- MMU
- IDTR
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
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
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.
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
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

operating system

hardware
Virtualisation

- process
- operating system
- hardware
Virtualisation

- process
- process

operating system

hardware
Virtualisation

- hardware
- operating system
- process
- process
- process
Virtualisation

- process
- operating system
- 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?
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?
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.
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
the Hypervisor

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
The operating system is running in non-privileged mode.

- 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 virtual IDT

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
handle interrupt
save ref to IDT of 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.

- **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
- continue as if nothing happened

The operating system is running in non-privileged mode.
<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>Guest operating system</th>
<th>Application</th>
<th>running</th>
</tr>
</thead>
</table>

*a system call*
<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>Guest operating system</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>running system call</td>
</tr>
</tbody>
</table>
a system call

<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></td>
<td></td>
<td>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

Hypervisor  Guest operating system  Application

running
system call
INT 0x80

handle interrupt
check OS IDT
A system call is a way for an application to interact with the underlying operating system. The process starts when the application requests a system call by executing the INT 0x80 instruction. This interrupt is handled by the hypervisor, which then checks the OS IDT (Interrupt Descriptor Table) to find the correct procedure to handle the request. The selected OS procedure is then called to execute the requested operation.
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

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

handle interrupt
check OS IDT
call OS procedure

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

11 / 24
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?

- process
- guest operating system
- hypervisor
- hardware

This will be expensive!
What about virtual memory?

- virtual addresses
- guest operating system
- hypervisor
- hardware

This will be expensive!
What about virtual memory?

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

- regular translation tables

This will be expensive!
What about virtual memory?

- virtual addresses
- regular translation tables
- second level translation
- 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>running</td>
<td></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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>page fault</td>
</tr>
</tbody>
</table>
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

handle interrupt  running page fault

call OS procedure
second level paging

Hypervisor  Guest operating system  Application

Running page fault

call OS procedure

handle interrupt
second level paging

Hypervisor → Guest operating system → Application

running page fault

handle interrupt
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

handle interrupt  call OS procedure  running page fault

map missing page
update page table
second level paging

Hypervisor → Guest operating system → Application

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

running page fault
second level paging

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

↓
handle interrupt  call OS procedure  running page fault

↑
map missing page  update page table

modify page table  return to OS
**second level paging**

A page fault occurs when an application attempts to access a page that is not currently in memory. The process flow is as follows:

1. **Hypervisor** handles the interrupt.
2. It calls an OS procedure.
3. The OS procedure maps the missing page.
4. It updates the page table.
5. Finally, it returns to the OS.
6. The OS returns to the user.
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

handle interrupt  call OS procedure  running page fault

map missing page  update page table

modify page table

call OS procedure  map missing page

return to OS  return to user

return to user  resume execution
...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 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.
<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</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>kernel space</td>
<td>user space</td>
<td>in user mode</td>
</tr>
<tr>
<td>system call</td>
<td></td>
<td>system call</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

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
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
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
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
system call revisited

Hypervisor

kernel space

change tables

OS now in user space

Guest operating system

user space

in user mode

system call

INT 0x80

Application

in user mode

handle interrupt

return to user
system call revisited

Hypervisor

kernel space

change tables

OS now in user space

Guest operating system

user space

in user mode

system call

INT 0x80

Application

in user mode

handle interrupt

return to user
system call revisited

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

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 | 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
  - OS in kernel space
  - Handle interrupt
    - Return to user
    - Change tables
  - OS now in user space
  - Change tables
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

return to user

change tables

OS in kernel space

resume execution
Hardware support needed - available in both AMD an Intel x86 processors.
Hardware support needed - available in both AMD an Intel x86 processors.

With hardware support, hypervisors can provide near “bare metal” performance.
the original goal
Utilisation of hardware.
Utilisation of hardware.
Utilisation of hardware.

Applications are completely separated from each other.
the original goal

Utilisation of hardware.

Applications are completely separated from each other.
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.

What if we skip this.
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.
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

hardware
containers

- process
- process
- process

- operating system

- hardware
containers

process

operating system

hardware
containers

process

operating system

hardware
containers

process

process

process

operating system

hardware
containers

operating system

hardware
the original goal

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?
Utilisation of hardware.
the original goal

Utilisation of hardware.
Utilisation of hardware.

Applications are completely separated from each other.
Utilisation of hardware.

Applications are completely separated from each other.
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.

Why do they have to run on the same hardware?
emulating hardware

x86 hardware
Hardware emulators can be surprisingly efficient.

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

- operating system
- x86 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.
Hardware emulators can be surprisingly efficient.
Hardware emulators can be surprisingly efficient.
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).
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.
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.