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

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.
Interruption - Asynchronous interrupts:
- timer interrupt
- hardware interrupt: I/O complete,
- ...

The kernel is interrupt driven.

Virtualisation

The Hypervisor

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
- 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.

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

The virtual IDT

<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>Guest Operating system</th>
</tr>
</thead>
<tbody>
<tr>
<td>set up IDT</td>
<td>initialize OS</td>
</tr>
<tr>
<td>pass control to OS</td>
<td>set up IDT</td>
</tr>
<tr>
<td>handle interrupt</td>
<td>continue as if nothing happened</td>
</tr>
<tr>
<td>save ref to IDT of OS</td>
<td></td>
</tr>
<tr>
<td>pass control to OS</td>
<td></td>
</tr>
</tbody>
</table>

The operating system is running in non-privileged mode.

a system call

<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>Guest operating system</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle interrupt</td>
<td></td>
<td>running system call</td>
</tr>
<tr>
<td>check OS IDT</td>
<td></td>
<td>INT 0x80</td>
</tr>
<tr>
<td>call OS procedure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>handle interrupt</td>
<td></td>
<td>return to user</td>
</tr>
<tr>
<td>return to user</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>resume execution</td>
</tr>
</tbody>
</table>

What about virtual memory?

- regular translation tables
- second level translation

This will be expensive!
User process uses virtual addresses that are automatically 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 returns to the user process.

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.
Thank God for Hardware

Hardware support needed - available in both AMD and Intel x86 processors.

With hardware support, hypervisors can provide near “bare metal” performance.

Applications are completely separated from each other.

Applications can use different operating systems.

What if we skip this.

Containers

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.
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?

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).

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.