6.033 Spring 2021

Lecture #6: Virtual Machines and Performance

wrapping up operating systems
**Operating Systems** enforce modularity on a single machine using virtualization in order to enforce modularity and have an effective operating system, a few things need to happen:

1. Programs shouldn’t be able to refer to (and corrupt) each others’ memory

2. Programs should be able to communicate with each other

3. Programs should be able to share a CPU without one program halting the progress of the others

- Virtual memory
- Bounded buffers (virtualize communication links)
- Threads (virtualize processors)
**operating systems** enforce modularity on a single machine using **virtualization** in order to enforce modularity + have an effective operating system, a few things need to happen

1. programs shouldn’t be able to refer to (and corrupt) each others’ **memory**

2. programs should be able to **communicate** with each other

3. programs should be able to **share a CPU** without one program halting the progress of the others

**today’s goal:** run multiple operating systems at once
**common computing environment:** running multiple OSes on a single physical machine

- **host OS**
  - virtual machine
    - running guest OS

- **physical hardware**
  - U/K, PTR, page table, …
**common computing environment:** running multiple OSes on a single physical machine

- virtual machine running guest OS
- virtual machine running guest OS
- physical hardware
  - U/K, PTR, page table, ...
common computing environment: running multiple OSes on a single physical machine

problem: how to (safely) share physical hardware?
virtual machine monitor: virtualizes the physical hardware for the guest OSes

virtual machine running guest OS

virtual machine running guest OS

virtual machine monitor (VMM)

physical hardware

U/K, PTR, page table, …
**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

guest OSes run in user mode
virtual machine monitor: virtualizes the physical hardware for the guest OSes.

guest OSes run in user mode.

privileged instructions in guest OS will cause an exception, which the VMM will intercept ("trap") and emulate. If the VMM can’t emulate an instruction, it will send the exception back to the guest OS for handling.
**virtual machine monitor**: virtualizes the physical hardware for the guest OSes

- **virtual machine running guest OS**
- **virtual machine running guest OS**
- **virtual machine monitor (VMM)**
- **physical hardware**: U/K, PTR, page table, ...

guest OSes run in user mode

privileged instructions in guest OS will cause an exception, which the VMM will intercept ("trap") and **emulate**

if the VMM can’t emulate an instruction, it will send the exception back to the guest OS for handling

**first question**: what does it mean to emulate?
virtual machine monitor: virtualizes the physical hardware for the guest OSes

guest OS

<table>
<thead>
<tr>
<th>virtual hardware</th>
<th>virtual hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>U/K</td>
<td>U/K</td>
</tr>
<tr>
<td>PTR</td>
<td>PTR</td>
</tr>
<tr>
<td>page table</td>
<td>page table</td>
</tr>
</tbody>
</table>

virtual machine monitor (VMM)

physical hardware

| U/K, PTR, page table, ... |

guest OSes run in user mode

privileged instructions in guest OS will cause an exception, which the VMM will intercept ("trap") and emulate

if the VMM can’t emulate an instruction, it will send the exception back to the guest OS for handling

first question: what does it mean to emulate?
**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again)

<table>
<thead>
<tr>
<th>guest OS</th>
<th>guest OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual hardware</td>
<td>virtual hardware</td>
</tr>
<tr>
<td>U/K</td>
<td>U/K</td>
</tr>
<tr>
<td>PTR</td>
<td>PTR</td>
</tr>
<tr>
<td>page table</td>
<td>page table</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>virtual machine monitor (VMM)</th>
<th>physical hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>U/K, PTR, page table, ...</td>
<td>U/K, PTR, page table, ...</td>
</tr>
</tbody>
</table>

**first question:** what does it mean to emulate?
**virtual machine monitor**: virtualizes the physical hardware for the guest OSes

First example: virtualizing memory (again)

<table>
<thead>
<tr>
<th>physical hardware</th>
<th>virtual hardware</th>
<th>virtual machine monitor (VMM)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U/K, PTR, page table...</td>
<td>U/K, PTR, page table...</td>
<td></td>
</tr>
</tbody>
</table>

**guest OS**

<table>
<thead>
<tr>
<th>virtual hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>U/K</td>
</tr>
<tr>
<td>PTR</td>
</tr>
<tr>
<td>page table</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

**guest virtual address**

**first question**: what does it mean to emulate?
**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

**first example:** virtualizing memory (again)

- **guest OS**
  - virtual hardware
    - U/K
    - PTR
    - page table
    - ...

- **guest OS**
  - virtual hardware
    - U/K
    - PTR
    - page table
    - ...

- **virtual machine monitor (VMM)**

- **physical hardware**
  - U/K, PTR, page table, ...

---

**first question:** what does it mean to emulate?
**virtual machine monitor**: virtualizes the physical hardware for the guest OSes

First example: virtualizing memory (again)

<table>
<thead>
<tr>
<th>virtual hardware</th>
<th>virtual hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>U/K, PTR, page table</td>
<td>U/K, PTR, page table</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>guest OS</th>
<th>guest OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>guest virtual address</td>
<td>guest physical address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>virtual machine monitor (VMM)</th>
<th>physical hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>U/K, PTR, page table, ...</td>
<td>U/K, PTR, page table, ...</td>
</tr>
</tbody>
</table>

**first question**: what does it mean to emulate?
**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

**first example:** virtualizing memory (again)

<table>
<thead>
<tr>
<th>guest OS</th>
<th>guest OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual hardware</td>
<td>virtual hardware</td>
</tr>
<tr>
<td>U/K, PTR, page table, ...</td>
<td>U/K, PTR, page table, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>virtual machine monitor (VMM)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>physical hardware</th>
<th>physical hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>U/K, PTR, page table, ...</td>
<td>U/K, PTR, page table, ...</td>
</tr>
</tbody>
</table>

**guest virtual address**

**guest physical address**

**host physical address**

**first question:** what does it mean to emulate?

In this example, it means that the VMM needs to step in and translate guest physical addresses to host physical addresses.
**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

First example: virtualizing memory (again)

```
<table>
<thead>
<tr>
<th>guest OS</th>
<th>guest OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual hardware</td>
<td>virtual hardware</td>
</tr>
<tr>
<td>guest OS page table</td>
<td>guest OS page table</td>
</tr>
</tbody>
</table>

| virtual machine monitor (VMM) |
| physical hardware |
```

PTR

Katrina LaCurts | lacurts@mit.edu | 6.033 2021
**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

First example: virtualizing memory (again)

<table>
<thead>
<tr>
<th>guest OS</th>
<th>guest OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual hardware</td>
<td>virtual hardware</td>
</tr>
<tr>
<td>guest OS page table</td>
<td>guest OS page table</td>
</tr>
<tr>
<td>PTR</td>
<td>PTR</td>
</tr>
</tbody>
</table>

---

**virtual machine monitor (VMM)**

**physical hardware**

---

guest virtual → guest physical
**virtual machine monitor**: virtualizes the physical hardware for the guest OSes

**first example: virtualizing memory (again)**

1. guest OS loads its PTR, which triggers an exception; the VMM intercepts
**virtual machine monitor**: virtualizes the physical hardware for the guest OSes

1. guest OS loads its PTR, which triggers an exception; the VMM intercepts
**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

First example: virtualizing memory (again)

1. guest OS loads its PTR, which triggers an exception; the VMM intercepts

2. VMM combines the guest page table with its own page table to create a host page table

---

**Diagram:**

- **guest OS**
  - virtual hardware
    - guest OS page table
    - PTR
  - virtual hardware
    - guest OS page table
    - PTR

- **virtual machine monitor (VMM)**
  - virtual hardware
    - guest OS page table

- **physical hardware**
  - PTR

---

Guest virtual $\rightarrow$ guest physical
**virtual machine monitor**: virtualizes the physical hardware for the guest OSes

**first example: virtualizing memory (again)**

1. guest OS loads its PTR, which triggers an exception; the VMM intercepts

2. VMM combines the guest page table with its own page table to create a host page table

---

guest OS

virtual hardware

- guest OS page table
- PTR

virtual hardware

- guest OS page table
- PTR

**virtual machine monitor (VMM)**

- guest OS page table
- VMM page table

physical hardware

- PTR
**Virtual Machine Monitor (VMM):** virtualizes the physical hardware for the guest OSes

**First Example: Virtualizing Memory (again)**

1. A guest OS loads its page table (PTR), which triggers an exception; the VMM intercepts.

2. The VMM combines the guest page table with its own page table to create a host page table.

---

**Diagram:**

- **Guest OS**
  - Virtual hardware
  - Guest OS page table
  - PTR

- **Guest OS**
  - Virtual hardware
  - Guest OS page table
  - PTR

- **Virtual Machine Monitor (VMM)**
  - Guest OS page table
  - VMM page table
  - Host page table

- **Physical Hardware**
  - PTR
virtual machine monitor: virtualizes the physical hardware for the guest OSes

first example: virtualizing memory (again)

1. guest OS loads its PTR, which triggers an exception; the VMM intercepts

2. VMM combines the guest page table with its own page table to create a host page table

3. physical hardware uses the host page table
**Virtual Machine Monitor (VMM):** virtualizes the physical hardware for the guest OSes.

First example: virtualizing memory (again)

1. Guest OS loads its PTR, which triggers an exception; the VMM intercepts.

In modern hardware, the physical hardware is aware of both page tables, and performs the translation from guest virtual to host physical itself.
**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

<table>
<thead>
<tr>
<th>guest OS</th>
<th>guest OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual hardware</td>
<td>virtual hardware</td>
</tr>
<tr>
<td>U/K</td>
<td>U/K</td>
</tr>
<tr>
<td>PTR</td>
<td>PTR</td>
</tr>
<tr>
<td>page table</td>
<td>page table</td>
</tr>
</tbody>
</table>

**virtual machine monitor (VMM)**

**physical hardware**

U/K, PTR, page table, …
**virtual machine monitor**: virtualizes the physical hardware for the guest OSes

<table>
<thead>
<tr>
<th>guest OS</th>
<th>guest OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual hardware</td>
<td>virtual hardware</td>
</tr>
<tr>
<td>U/K, PTR, page table, …</td>
<td>U/K, PTR, page table, …</td>
</tr>
</tbody>
</table>

**virtual machine monitor (VMM)**

<table>
<thead>
<tr>
<th>physical hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>U/K, PTR, page table, …</td>
</tr>
</tbody>
</table>

guest OSes run in user mode
**Virtual Machine Monitor (VMM):** virtualizes the physical hardware for the guest OSes.

- **Guest OS**
  - **Virtual Hardware**
    - U/K
    - PTR
    - Page table
    - ...
  - **Virtual Machine Monitor (VMM)**
  - **Physical Hardware**
    - U/K, PTR, Page table, ...

Guest OSes run in user mode.

Privileged instructions in guest OS will cause an exception, which the VMM will intercept ("trap") and emulate.

If the VMM can't emulate an instruction, it will send the exception back to the guest OS for handling.
**virtual machine monitor**: virtualizes the physical hardware for the guest OSes

<table>
<thead>
<tr>
<th>guest OS</th>
<th>guest OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual hardware</td>
<td>virtual hardware</td>
</tr>
<tr>
<td>U/K, PTR, page table, ...</td>
<td>U/K, PTR, page table, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>virtual machine monitor (VMM)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>physical hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>U/K, PTR, page table, ...</td>
</tr>
</tbody>
</table>

**guest OSes run in user mode**

privileged instructions in guest OS will cause an exception, which the VMM will intercept (“**trap**”) and **emulate**

if the VMM can’t emulate an instruction, it will send the exception back to the guest OS for handling

**second question**: what about when the VMM needs to emulate an instruction that doesn’t trigger an exception?
**virtual machine monitor**: virtualizes the physical hardware for the guest OSes

second example: virtualizing the U/K bit

<table>
<thead>
<tr>
<th>guest OS</th>
<th>guest OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual hardware</td>
<td>virtual hardware</td>
</tr>
<tr>
<td>U/K, PTR, page table, ...</td>
<td>U/K, PTR, page table, ...</td>
</tr>
<tr>
<td>virtual machine monitor (VMM)</td>
<td>virtual machine monitor (VMM)</td>
</tr>
<tr>
<td>physical hardware</td>
<td>physical hardware</td>
</tr>
<tr>
<td>U/K, PTR, page table, ...</td>
<td>U/K, PTR, page table, ...</td>
</tr>
</tbody>
</table>

**second question**: what about when the VMM needs to emulate an instruction that doesn’t trigger an exception?
**virtual machine monitor:** virtualizes the physical hardware for the guest OSes

second example: virtualizing the U/K bit

---

<table>
<thead>
<tr>
<th></th>
<th>guest OS</th>
<th>guest OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual hardware</td>
<td></td>
<td></td>
</tr>
<tr>
<td>U/K</td>
<td>U/K</td>
<td></td>
</tr>
<tr>
<td>PTR</td>
<td>PTR</td>
<td></td>
</tr>
<tr>
<td>page table</td>
<td>page table</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>virtual hardware</th>
<th>virtual hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>U/K</td>
<td>U/K</td>
<td>U/K</td>
</tr>
<tr>
<td>PTR</td>
<td>PTR</td>
<td>PTR</td>
</tr>
<tr>
<td>page table</td>
<td>page table</td>
<td>page table</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>virtual machine monitor (VMM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>physical hardware</td>
<td>U/K, PTR, page table, ...</td>
</tr>
</tbody>
</table>

**para-virtualization:** modify guest OS slightly

---

**second question:** what about when the VMM needs to emulate an instruction that doesn’t trigger an exception?
**virtual machine monitor**: virtualizes the physical hardware for the guest OSes

second example: virtualizing the U/K bit

<table>
<thead>
<tr>
<th>guest OS</th>
<th>guest OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual hardware</td>
<td>virtual hardware</td>
</tr>
<tr>
<td>U/K, PTR, page table, ...</td>
<td>U/K, PTR, page table, ...</td>
</tr>
</tbody>
</table>

**para-virtualization**: modify guest OS slightly

**binary translation**: VMM replaces problematic instructions with ones that it can trap and emulate

**second question**: what about when the VMM needs to emulate an instruction that doesn’t trigger an exception?
virtual machine monitor: virtualizes the physical hardware for the guest OSes

second example: virtualizing the U/K bit

<table>
<thead>
<tr>
<th>guest OS</th>
<th>guest OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual hardware</td>
<td>virtual hardware</td>
</tr>
<tr>
<td>U/K</td>
<td>U/K</td>
</tr>
<tr>
<td>PTR</td>
<td>PTR</td>
</tr>
<tr>
<td>page table</td>
<td>page table</td>
</tr>
</tbody>
</table>

virtual machine monitor (VMM)

physical hardware

U/K, PTR, page table, ...

para-virtualization: modify guest OS slightly

binary translation: VMM replaces problematic instructions with ones that it can trap and emulate

hardware support: architecture provides a special operating mode for VMMs in addition to user mode, kernel mode

second question: what about when the VMM needs to emulate an instruction that doesn’t trigger an exception?
**monolithic kernel:** no enforced modularity within the kernel itself

- Application
- Basic interprocess communication, virtual memory, scheduling, file server, device drivers, network, ...
- Hardware
**monolithic kernel**: no enforced modularity within the kernel itself

- Application
- Basic interprocess communication, virtual memory, scheduling, file server, device drivers, network, ...
- Hardware

**microkernels**: enforce modularity by putting subsystems in user programs

- Application
- Basic interprocess communication, virtual memory, scheduling
- Hardware

- Application
- IPC
- Device driver
- Network
- ...
- Basic interprocess communication, virtual memory, scheduling
- Hardware
**Performance:** performance issues have influenced a lot of the system designs you’ve seen so far.
**performance:** performance issues have influenced a lot of the system designs you’ve seen so far

**latency:** how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of memory in an OS?
**performance:** performance issues have influenced a lot of the system designs you’ve seen so far

**latency:** how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of memory in an OS?

**throughput:** how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?
**performance:** performance issues have influenced a lot of the system designs you’ve seen so far

**latency:** how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of memory in an OS?

**throughput:** how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?
**performance**: performance issues have influenced a lot of the system designs you’ve seen so far

**latency**: how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of memory in an OS?

**throughput**: how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?

often we see latency remain low until the system is heavily utilized, and then it rises.

throughput, on the other hand, increases as requests increase, and maxes out when the system is fully utilized.
**performance**: performance issues have influenced a lot of the system designs you’ve seen so far

**latency**: how long does it take to complete a single request?

example: how long does it take to retrieve a particular piece of memory in an OS?

**throughput**: how many requests per unit of time?

example: how many reads or writes can a system do to a disk at once?

**utilization**: what fraction of resources are being utilized? this puts our performance measurements in context

often we see latency remain low until the system is heavily utilized, and then it rises.

throughput, on the other hand, increases as requests increase, and maxes out when the system is fully utilized.
**virtual machines** allow us to run multiple **isolated** OSes on a single physical machine, similar to how we used an OS to run multiple programs on a single CPU.
Virtual machines allow us to run multiple isolated OSes on a single physical machine, similar to how we used an OS to run multiple programs on a single CPU.

Monolithic kernels provide no enforced modularity within the kernel. Microkernels do, but redesigning monolithic kernels as microkernels is challenging.
**virtual machines** allow us to run multiple **isolated** OSes on a single physical machine, similar to how we used an OS to run multiple programs on a single CPU.

**monolithic kernels** provide no enforced modularity within the kernel. **microkernels** do, but redesigning monolithic kernels as microkernels is challenging.

**performance** affects all aspects of system design. **throughput**, **latency**, and **utilization** are important metrics (but not the only ones), and we have general, systems-level techniques for attempting to improve them (e.g., caching, batching). knowing when to apply them requires a solid understanding of the system itself and how it’s used.