1. Virtualization basics

2. Hardware assisted virtualization

3. Example: Mini VMM with KVM

4. I/O virtualization
   - How do modern Network Interface Cards (NIC) work
   - Device emulation
   - Virtio
   - PCI pass-through
   - Single-Root I/O Virtualization
   - Inter-VM networking

5. Summary
Outline

1. Virtualization basics
2. Hardware assisted virtualization
3. Example: Mini VMM with KVM
4. I/O virtualization
   - How do modern Network Interface Cards (NIC) work
   - Device emulation
   - Virtio
   - PCI pass-through
   - Single-Root I/O Virtualization
   - Inter-VM networking
5. Summary
Virtualization basics

Virtualization

- **Definition:** Virtualization of the whole computing platform – the operating system thinks it runs on real hardware, but the hardware is largely emulated by hypervisor and/or virtual machine monitor (VMM).

- **Virtual machine (VM) vs. Java VM**
  - Java VM interprets Java byte code and interacts with an operating system
  - VM executes native (machine) code and interacts with a hypervisor.

- **Used since ’70, mostly on IBM mainframes**
  - Popek and Goldberg defined requirements for ISA virtualization in their paper in 1974,
  - x86 became fully virtualizable in 2005.

- **More detailed introduction to virtualization (from OSY course):**
**Basic mechanism of virtualization**

- **Sensitive instruction**: Changes *global state*\(^1\) or behaves differently depending on *global state* (e.g. cli, pushf on x86)
- **Privileged instruction**: Unprivileged execution **traps** to the privileged mode (hypervisor, CPU exception)

- on x86 `popf`, `pushf` and few other instructions were not privileged!
  - `pushf` stores all flags to stack (including “global” interrupt flag)
  - `popf` sets IF in privileged mode and ignores it in unprivileged mode (does not trap)

- **Hypervisor (HV)** can **emulate** the effect of sensitive instructions depending on the VM state (not the global state).

\(^1\) Global state means a state that is common to all running VMs, not local to a single VM. For example, CPU reset signal is global.
Privileged code that supervises execution of the VM, i.e. handles traps.

Hypervisor types:

- Bare-metal hypervisor (Xen, VMware ESX, ...)
- Hosted hypervisor (KVM, VirtualBox)

The boundary is blurry – many bare-metal hypervisors support native apps
Software that emulates HW platform (network, graphics, storage, ...)
- Often implemented inside hypervisor (left) ⇒ people confuse VMM with hypervisors
- Today’s platforms are complex (e.g. PC bears 40 years heritage)
- It is more secure to execute the VMM in user mode, outside of privileged mode (right, example: KVM & qemu)
- It is also slower, but see NOVA microhypervisor (TU Dresden), which implements this faster.
Questions

- How many privilege levels we need to implement virtualization?
  - Two are sufficient, but then, every guest system call, page fault etc. traps from the guest app to the hypervisor, which then安排s switch to guest kernel – slow.
  - Hardware assisted virtualization – introduces more privilege levels and more – see later.

- Why is virtualization needed at all? (My personal rant)
  - To some extent because the design of mainstream operating systems is not up to the current needs.
  - Current OSes do not offer sufficient isolation of applications and groups of applications. Many things such as user permissions, apply implicitly to the whole system.
  - Microkernel OSes, which solve this problem, were designed in the past without much success.
  - Now, people are adding “containers” to mainstream OSes, which is painful and often with security problems.
    - Making a microkernel from a monolithic kernel is more difficult that starting with microkernel from scratch.
Outline

1. Virtualization basics
2. Hardware assisted virtualization
3. Example: Mini VMM with KVM
4. I/O virtualization
   - How do modern Network Interface Cards (NIC) work
   - Device emulation
   - Virtio
   - PCI pass-through
   - Single-Root I/O Virtualization
   - Inter-VM networking
5. Summary
Hardware assisted virtualization

- Accelerates virtualized execution
- Differences between vendors (Intel, AMD, ARM, ...), core principles similar:
  - More privilege levels (x86 – root/non-root, ARMv8 EL0–3)
  - Nested paging
  - IO virtualization
Hardware assisted virtualization

Intel VMX

- VMX root operation
  (host rings 0–3)
- VMX non-root operation
  (guest rings 0–3)
- root → non-root = **VM Enter**
  - instructions: vmlaunch, vmresume
- non-root → root = **VM Exit**
  - instructions: vmresume, vmcall
  - faults (e.g. I/O)

- VM Control Structure (VMCS)
  - Data structure in memory that controls VMX execution (managed by hypervisor/VMM)
  - (Re)stores host/guest state
  - “Large structure” ⇒ VM Enter/Exit has overhead
  - The overhead depends on what is (re)stored from/to VMCS (configurable)

**VMCS (up to 4 KiB – e.g. 1024 B)**

<table>
<thead>
<tr>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM-execution control fields</td>
</tr>
<tr>
<td>Host state</td>
</tr>
<tr>
<td>Guest state</td>
</tr>
<tr>
<td>VM-exit information fields</td>
</tr>
<tr>
<td>VM-entry control fields</td>
</tr>
<tr>
<td>VM-exit control fields</td>
</tr>
</tbody>
</table>
Nested paging & address spaces

- Host physical
- Guest physical
- Guest virtual
- Host virtual
- Host page tables
- Guest page tables
- Hypervisor
- PDPT
- PD
- PT
- HW
- Guest
Memory access overhead

- TLB misses and page faults are more expensive in a VM!
- Page walk in a VM (worst case):
  1. Translate PDPT (CR3) address using host page tables (3 memory accesses for 3-level page tables)
  2. Translate PD address using host page tables (3 accesses)
  3. Translate PT address using host page tables (3 accesses)
- Performance drop up to 15/38% (Intel/AMD)\(^2\)
- Tagged TLBs
  - No need to flush TLBs on process (or VM) switches (good)
  - Applications share TLBs with hypervisor and VMM (bad)
- Recommendation: Use huge pages if possible

\(^2\)Ulrich Drepper, The Cost of Virtualization, ACM Queue, Vol. 6 No. 1 – 2008
Outline

1. Virtualization basics
2. Hardware assisted virtualization
3. Example: Mini VMM with KVM
4. I/O virtualization
   - How do modern Network Interface Cards (NIC) work
   - Device emulation
   - Virtio
   - PCI pass-through
   - Single-Root I/O Virtualization
   - Inter-VM networking
5. Summary
KVM

- Linux-based hosted hypervisor
- Abstracts hardware-assisted virtualization of different architectures behind `ioctl`-based API
- We will develop a miniature user-space VMM
  - Simplest hardware to virtualize: serial port

1. Setup the VM's memory
2. Load the code to execute
3. Run the VM
4. Handle the VM Exits and emulate serial port
5. Goto 3

See also https://lwn.net/Articles/658511/
Outline

1. Virtualization basics
2. Hardware assisted virtualization
3. Example: Mini VMM with KVM
4. I/O virtualization
   - How do modern Network Interface Cards (NIC) work
   - Device emulation
   - Virtio
   - PCI pass-through
   - Single-Root I/O Virtualization
   - Inter-VM networking
5. Summary
Outline

1. Virtualization basics
2. Hardware assisted virtualization
3. Example: Mini VMM with KVM
4. I/O virtualization
   - How do modern Network Interface Cards (NIC) work
   - Device emulation
   - Virtio
   - PCI pass-through
   - Single-Root I/O Virtualization
   - Inter-VM networking
5. Summary
I/O virtualization » How do modern Network Interface Cards (NIC) work

Network Interface Card & transmit operation

NIC Registers

Buffer descriptors (in memory)

Packet data

TX operation:
1. Write packet data
2. Fill in empty buffer descriptor
3. Notify NIC by writing TX tail reg
**Network Interface Card & receive operation**

**RX operation:**

1. Allocate packet buffers and update buffer descriptors
2. Update RX head/tail regs
3. On packet RX, NIC generates an interrupt
I/O virtualization » How do modern Network Interface Cards (NIC) work

Network Interface Card & SG DMA

Scatter-Gather DMA:

- Final packet is composed from several pieces scattered in memory
- Typically header (from OS) and data (from app)
Outline

1. Virtualization basics
2. Hardware assisted virtualization
3. Example: Mini VMM with KVM
4. I/O virtualization
   - How do modern Network Interface Cards (NIC) work
   - Device emulation
     - Virtio
     - PCI pass-through
     - Single-Root I/O Virtualization
     - Inter-VM networking
5. Summary
NIC device emulation

- Trap accesses to NIC registers (memory-mapped IO)
- Upon write to TX tail, VMM iterates over queued buffers and sends them via real NIC (e.g. SOCK_RAW)
- Multiple packets can be sent during single VM Exit (⇒ less overhead)
- Reception works similarly

- Not all hardware is “that nice” to virtualize
- Several VM Exits per TX or RX
- Registers that must be trapped are intermixed with non-sensitive (e.g. read-only) registers in a single page
  - ⇒ Unnecessary VM Exits for some register accesses
- VMM must emulate not only RX/TX, but also management
  - Link negotiation, configuration, ...
  - More complex compared to RX/TX
Outline

1. Virtualization basics
2. Hardware assisted virtualization
3. Example: Mini VMM with KVM
4. I/O virtualization
   - How do modern Network Interface Cards (NIC) work
   - Device emulation
   - Virtio
   - PCI pass-through
   - Single-Root I/O Virtualization
   - Inter-VM networking
5. Summary
Virtio

- It is neither easy nor necessary to emulate a real NIC
- TX, RX and simple configuration (e.g. MAC address) is sufficient
- Why to implement different ring-buffer formats?

Virtio\(^3\)

- Universal ring-buffer-based communication between VM and HV
- Used for network, storage, serial line, ...
- PCI-based probing & configuration – VMs can easily discover virtio devices

---

Outline

1. Virtualization basics
2. Hardware assisted virtualization
3. Example: Mini VMM with KVM
4. I/O virtualization
   - How do modern Network Interface Cards (NIC) work
   - Device emulation
   - Virtio
   - PCI pass-through
   - Single-Root I/O Virtualization
   - Inter-VM networking
5. Summary
I/O virtualization » PCI pass-through

PCI pass-through

- Even virtio needs one VM Exit per (a batch of) TX operation(s)
- If we don’t want VM Exits, we may want to give a VM exclusive access to the NIC
- Few problems to solve...
PCI pass-through graphically
PCI pass-through

Problems:

1. Virtual address space (see previous slide)
   - Security: One VM could configure the NIC to read or write memory of other VM or even the hypervisor!

2. Device interrupts
   - Host does not know how to acknowledge (silence) the interrupt – it has no driver for the device
   - It injects interrupt to the VM and returns from IRQ handler
   - Host is interrupted again, because VM didn’t have chance to run and ack the interrupt

Solution: Hardware support for direct use of devices in VMs

1. IOMMU (AMD), VT-d (Intel), SMMU (ARM)
2. Mask individual sources of interrupts without understanding the device
   - Hard with PCI, where interrupt lines are shared between devices
   - Possible with Message Signaled Interrupts (MSI)
Outline

1. Virtualization basics
2. Hardware assisted virtualization
3. Example: Mini VMM with KVM
4. I/O virtualization
   - How do modern Network Interface Cards (NIC) work
   - Device emulation
   - Virtio
   - PCI pass-through
   - Single-Root I/O Virtualization
   - Inter-VM networking
5. Summary
PCI pass-through is nice, but I have more VMs that want to communicate...
- Each VM has emulated NIC, VMM multiplexes the real NIC between VMs in software
- ... or perform the multiplexing in hardware

SR-IOV
- Besides “classic” physical function (PF), NIC implements several virtual functions (VFs)
- Each VF provides simplified PCI interface and its own RX/TX ring buffers
Outline

1 Virtualization basics
2 Hardware assisted virtualization
3 Example: Mini VMM with KVM
4 I/O virtualization
   ▪ How do modern Network Interface Cards (NIC) work
   ▪ Device emulation
   ▪ Virtio
   ▪ PCI pass-through
   ▪ Single-Root I/O Virtualization
   ▪ Inter-VM networking
5 Summary
Packet stored in VM’s memory
- VMM notified (VM Exit) e.g. via virtio’s kick()
- VMM notifies the SW switch via standard IPC mechanism
- Switch does memcpy() of the packet from source VM to destination VM (into dest NIC ring buffer)
- Dest VMM notifies the VM (injects interrupt)
Optimizations

- OS networking stack is responsible for splitting application data to packets (e.g. TCP segmentation) and adding appropriate headers.
- VMM sees many small packets and switch does many small memcpy()s.
- Receiver’s networking stack strips packet headers and combines the payload to larger data chunks for application.

- Segmentation is not necessary for Inter-VM communication (overhead)!
- Modern NICs support TCP Segmentation Offload (TSO)/Large Receive Offload (LRO): Segmentation/reconstruction is done in hardware.
- If virtual NIC supports TSO/LRO, Inter-VM communication is much faster, because whole TCP segments (in contrast to small packets) can be copied at once.
Outline

1 Virtualization basics
2 Hardware assisted virtualization
3 Example: Mini VMM with KVM

4 I/O virtualization
   ▪ How do modern Network Interface Cards (NIC) work
   ▪ Device emulation
   ▪ Virtio
   ▪ PCI pass-through
   ▪ Single-Root I/O Virtualization
   ▪ Inter-VM networking

5 Summary
Virtualization is just “another layer of indirection” and as such it adds overheads.

It is useful to know where the overheads are and how to mitigate them.