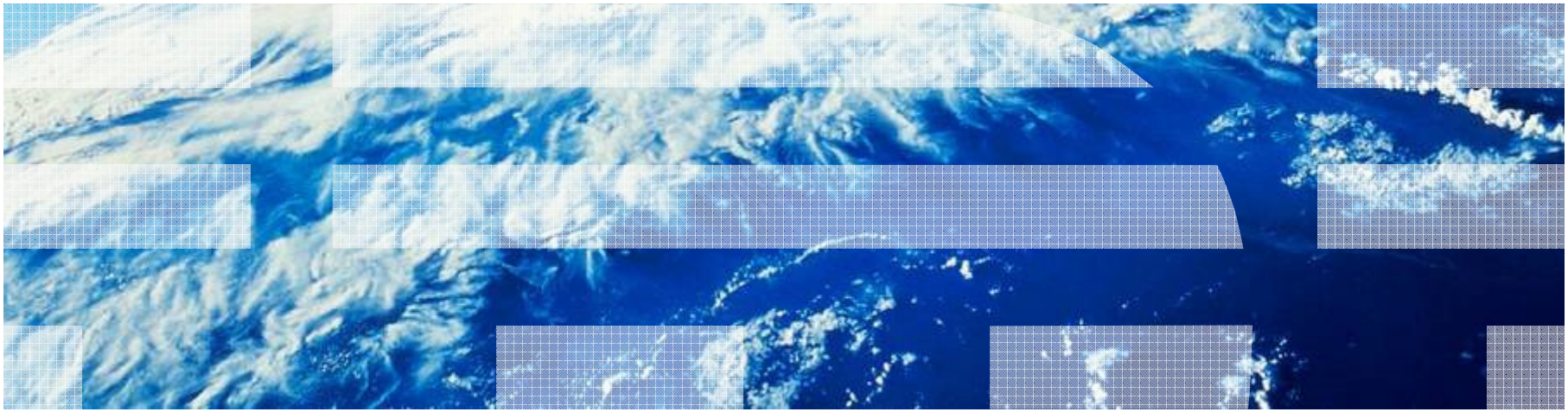
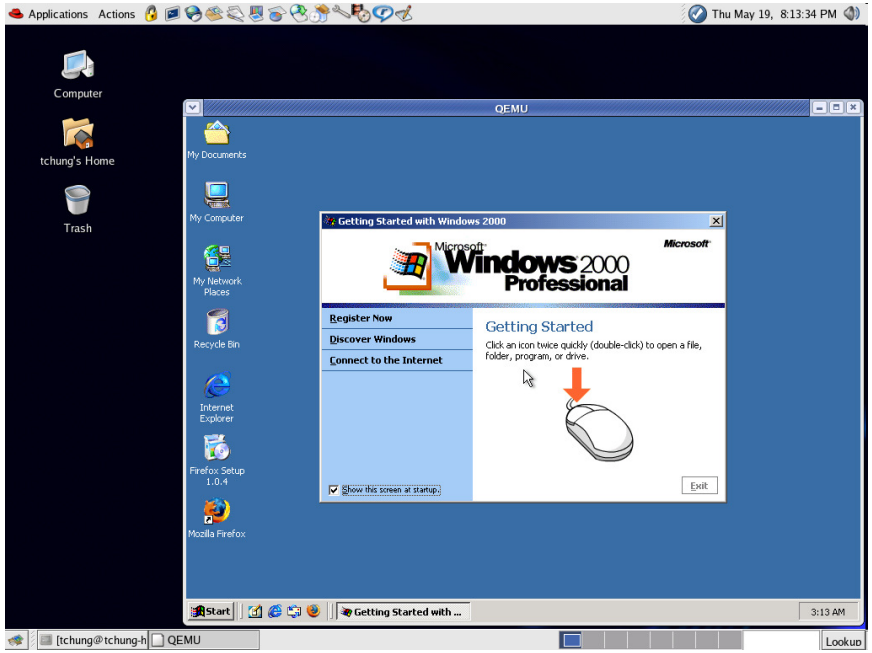
Virtualization Technologies

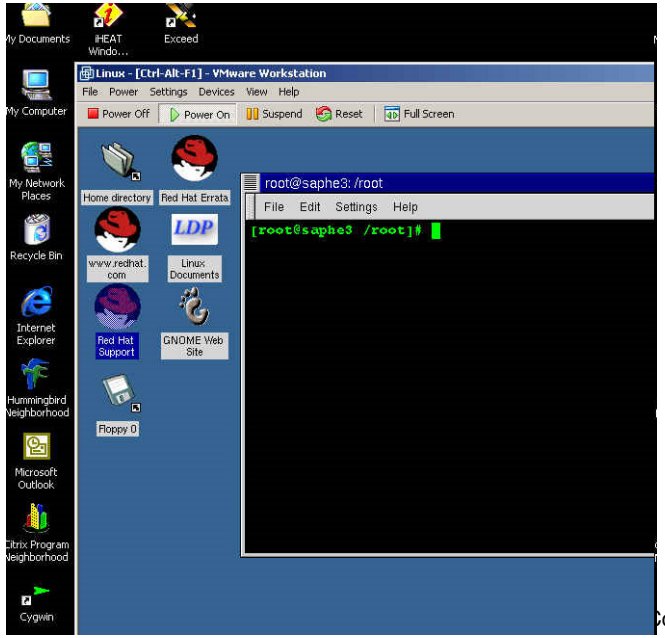
Alex Landau (lalex@il.ibm.com)
IBM Haifa Research Lab



What is virtualization?

- Virtualization is way to run **multiple operating systems** and **user applications** on the same hardware
  - E.g., run both Windows and Linux on the same laptop
- How is it different from **dual-boot**?
  - Both OSes run **simultaneously**
- The OSes are completely **isolated** from each other





Uses of virtualization

- Server consolidation
  - Run a **web server** and a **mail server** on the **same physical server**
- Easier development
  - Develop critical **operating system components** (file system, disk driver) without affecting **computer stability**
- QA
  - Testing a network product (e.g., a firewall) may require **tens of computers**
  - Try testing thoroughly a product at each pre-release milestone… and have a straight face when your boss shows you the **electricity bill**
- Cloud computing
  - The modern buzz-word
  - Amazon sells computing power
  - You pay for e.g., 2 CPU cores for 3 hours plus 10GB of network traffic

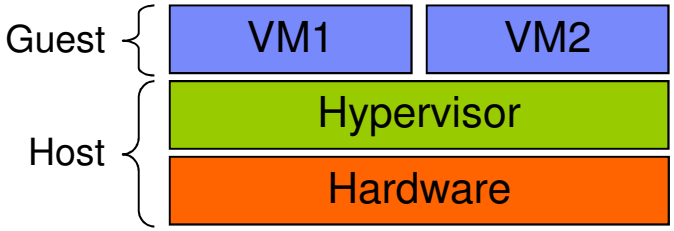
What's new in that? We've been doing it for decades!

- Indeed – an OS provides **isolation** between **processes**
  - Each has it's own **virtual memory**
  - Controlled access to **I/O devices** (disk, network) via system calls
  - Process **scheduler** to decide which process runs on which CPU core
- So what's the hype about?
- Try running Microsoft Exchange requiring **Windows and** your internal warehouse mgmt. application requiring **Linux simultaneously** on the same server!
- Or better yet, try to persuade **competing companies** to run their **processes side-by-side** in Amazon's **cloud** (had it not been virtualized)
- Psychological effect – what sounds better?
  - You're given **your own virtual machine** and you're **root** there – do whatever you want
  - You can run **certain processes**, but you **don't get root**, call our helpdesk with your configuration requests and we'll get back to you in 5 business days…

Two types of hypervisors

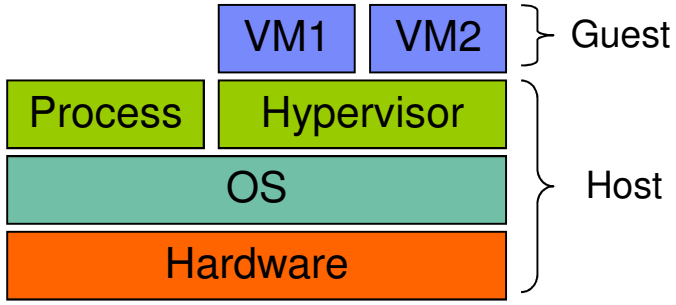
- Definitions
  - **Hypervisor** (or **VMM** – Virtual Machine Monitor) is a software layer that allows several **virtual machines** to **run** on a **physical machine**
  - The physical OS and hardware are called the **Host**
  - The virtual machine OS and applications are called the **Guest**

Type 1 (bare-metal)



VMware ESX, Microsoft Hyper-V, Xen

Type 2 (hosted)



VMware Workstation, Microsoft Virtual PC, Sun VirtualBox, QEMU, KVM

Bare-metal or hosted?

- **Bare-metal**
  - Has complete **control over hardware**
  - Doesn't have to **"fight" an OS**
- **Hosted**
  - Avoid **code duplication**: need not code a **process scheduler**, **memory management** system – the **OS already does** that
  - Can run native **processes alongside** VMs
  - Familiar environment – **how much CPU** and **memory** does a VM take? Use **top**! How big is the **virtual disk**? **ls –l**
  - Easy management – stop a VM? Sure, just kill it!
- A combination
  - Mostly hosted, but some parts are inside the OS kernel for performance reasons
  - E.g., **KVM**

How to run a VM? Emulate!

- Do whatever the **CPU** does but in **software**
- **Fetch** the next instruction
- **Decode** – is it an ADD, a XOR, a MOV?
- **Execute** – using the emulated registers and memory

Example:

addl %ebx, %eax

is emulated as:

enum {EAX=0, EBX=1, ECX=2, EDX=3, …};

unsigned long regs[8];

regs[EAX] += regs[EBX];

How to run a VM? Emulate!

- Pro:
  - Simple!
- Con:
  - Slooooooooow
- Example hypervisor: BOCHS

How to run a VM? Trap and emulate!

- Run the VM **directly** on the CPU – **no emulation**!
- Most of the code can execute just fine
  - E.g., addl %ebx, %eax
- **Some code** needs **hypervisor** intervention
  - int $0x80
  - movl something, %cr3
  - I/O
- Trap and emulate it!
  - E.g., if guest runs int $0x80, **trap** it and **execute guest's** interrupt 0x80 **handler**

How to run a VM? Trap and emulate!

- Pro:
  - Performance!

- Cons:
  - **Harder** to implement
  - Need **hardware support**
    - Not all "**sensitive**" instructions cause a **trap** when executed in usermode
    - E.g., **POPF**, that may be used to clear IF
    - This instruction does **not trap**, but value of **IF does not change**!
  - This hardware support is called VMX (Intel) or SVM (AMD)
  - Exists in modern CPUs

- Example hypervisor: KVM

How to run a VM? Dynamic (binary) translation!

- Take a block of binary VM code that is about to be executed
- Translate it **on the fly** to "safe" code (like JIT – just in time compilation)
- **Execute** the new **"safe" code directly** on the CPU

- Translation rules?
  - Most code translates **identically** (e.g., movl %eax, %ebx translates to itself)
  - "**Sensitive**" operations are translated into **hypercalls**
    - **Hypercall** – call into the hypervisor to ask for service
    - Implemented as **trapping** instructions (unlike POPF)
    - Similar to **syscall** – call into the OS to request service

How to run a VM? Dynamic (binary) translation!

- Pros:
  - No **hardware** support required
  - **Performance** – better than emulation
- Cons:
  - **Performance** – worse than trap and emulate
  - **Hard** to implement – hypervisor needs **on-the-fly** x86-to-x86 binary **compiler**
- Example hypervisors: VMware, QEMU

How to run a VM? Paravirtualization!

- Does **not run unmodified** guest OSes
- Requires **guest** OS to "**know**" it is running on top of a **hypervisor**

- E.g., instead of doing **cli** to turn off interrupts, guest OS should do **hypercall(DISABLE_INTERRUPTS)**

How to run a VM? Paravirtualization!

- Pros:
  - No **hardware** support required
  - **Performance** – better than emulation
- Con:
  - Requires **specifically modified guest**
  - Same guest OS cannot run in the VM and bare-metal
- Example hypervisor: Xen

Industry trends

- Trap and emulate
- With hardware support
- VMX, SVM

I/O Virtualization

- We saw **methods** to **virtualize** the **CPU**
- A computer is more than a CPU
- Also need I/O!

- Types of I/O:
  - Block (e.g., hard disk)
  - Network
  - Input (e.g., keyboard, mouse)
  - Sound
  - Video
- Most performance critical (for servers):
  - Network
  - Block

Side note – How does a NIC (network interface card) driver work?

- Transmit path:
  - OS prepares packet to transmit in a buffer in memory
  - Driver writes **start address** of buffer to **register X** of the NIC
  - Driver writes **length** of buffer to **register Y**
  - Driver writes '1' (**GO!**) into **register T**
  - NIC reads packet from memory addresses [X,X+Y) and sends it on the wire
  - NIC sends interrupt to host (**TX complete**, next packet please)
- Receive path:
  - Driver prepares buffer to receive packet into
  - Driver writes **start address** of buffer to **register X**
  - Driver writes **length** of buffer to **register Y**
  - Driver writes '1' (**READY-TO-RECEIVE**) into **register R**
  - When packet arrives, NIC copies it into memory at [X,X+Y)
  - NIC interrupts host (**RX**)
  - OS processes packet (e.g., wake the waiting process up)

I/O Virtualization? Emulate!

- Hypervisor implements **virtual NIC** (by the specification of a real NIC, e.g., Intel, Realtek, Broadcom)
- NIC **registers** (X, Y, Z, T, R, …) are just **variables** in hypervisor (host) **memory**
- If **guest writes** ‘1’ to **register T**, **hypervisor reads** buffer from **memory [X,X+Y)** and passes it to **physical NIC** driver for transmission
- When physical NIC interrupts (**TX complete**), hypervisor **injects** TX complete interrupt into guest

- Similar for RX path

I/O Virtualization? Emulate!

- Pro:
  - **Unmodified guest** (guest already has drivers for Intel NICs…)
- Cons:
  - **Slow** – every access to every NIC register causes a **VM exit** (trap to hypervisor)
  - Hypervisor needs to **emulate complex hardware**
- Example hypervisors: QEMU, KVM, VMware (without VMware Tools)

I/O Virtualization? Paravirtualize!

- Add virtual NIC driver into guest (**frontend**)
- Implement the virtual NIC in the hypervisor (**backend**)
- Everything works just like in the emulation case…
- …except – **protocol** between frontend and backend

- Protocol in emulation case:
  - Guest writes registers X, Y, waits at least 3 nano-sec and writes to register T
  - Hypervisor **infers** guest wants to transmit packet
- Paravirtual protocol:
  - Guest does a hypercall, passes it start address and length as arguments
  - Hypervisor **knows** what it should do
- Paravirtual protocol can be **high-level**, e.g., ring of buffers to transmit (so NIC doesn't stay idle after one transmission), and **independent of particular NIC** registers

I/O Virtualization? Paravirtualize!

- Pro:
  - **Fast** – no need to emulate physical device
- Con:
  - Requires **guest driver**
- Example hypervisors: QEMU, KVM, VMware (with VMware Tools), Xen
- How is paravirtual I/O different from paravirtual guest?
  - Paravirtual guest requires to modify **whole OS**
    - Try doing it on Windows (without source code), or even Linux (lots of changes)
  - Paravirtual I/O requires the addition of a **single driver** to a guest
    - Easy to do on both Windows and Linux guests

I/O Virtualization? Direct access / direct assignment!

- “Pull” NIC out of the host, and “plug” it into the guest
- Guest is allowed to access NIC registers **directly**, no hypervisor intervention
- Host can’t access NIC anymore

I/O Virtualization? Direct access / direct assignment!

- Pro:
  - As **fast** as possible!
- Cons:
  - Need NIC per guest
  - Plus one for host
  - Can't do "cool stuff"
    - Encapsulate guest packets, monitor, modify them at the hypervisor level
- Example hypervisors: KVM, Xen, VMware

I/O Virtualization? Emerging standard – SR-IOV!

- Single root I/O virtualization
- Contains a **physical function** controlled by the host, used to create **virtual functions**
- Each virtual function is assigned to a guest (like in **direct assignment**)
- Each **guest thinks** it has **full control** of NIC, accesses registers directly
- NIC does multiplexing/demultiplexing of traffic

I/O Virtualization? Emerging standard – SR-IOV!

- Pros:
  - As **fast** as possible!
  - Need only one NIC (as opposed to direct assignment)
- Cons:
  - Emerging standard
    - Few hypervisors fully support it
    - Expensive!
    - Requires new hardware
  - Can't do "cool stuff"
- Example hypervisors: KVM, Xen, VMware

Industry trends on I/O virtualization

- SR-IOV is the fastest
- Also, the most expensive

- Paravirtual I/O is cheap
- But much worse performance

- Companies (Red Hat, IBM, …) are looking at paravirtual I/O, trying to optimize it

- Winner still unknown

Advanced topics

- Memory over-commit
- Nested virtualization
- Live migration

The end!

Questions?

Alex Landau

lalex@il.ibm.com