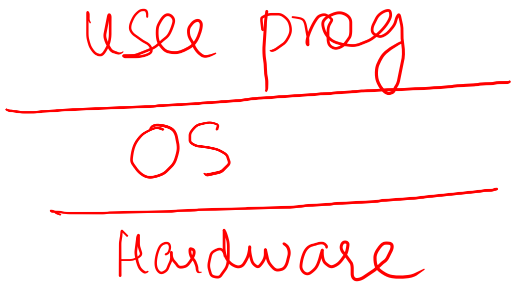
Design and Engineering of Computer Systems

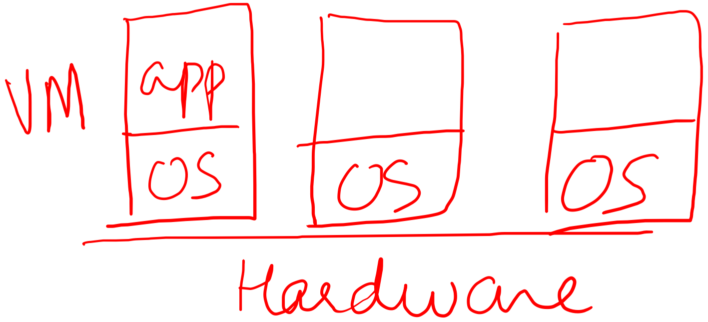
Lecture 10: Virtual machines and containers

Mythili Vutukuru

IIT Bombay

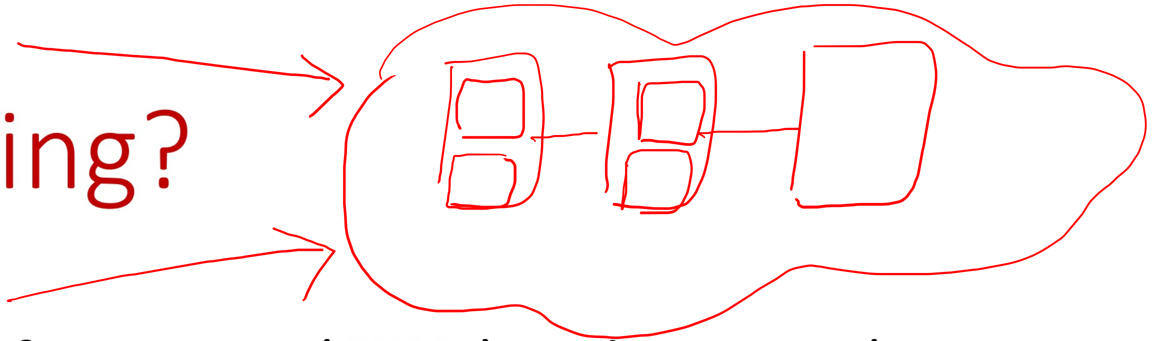
Virtualization





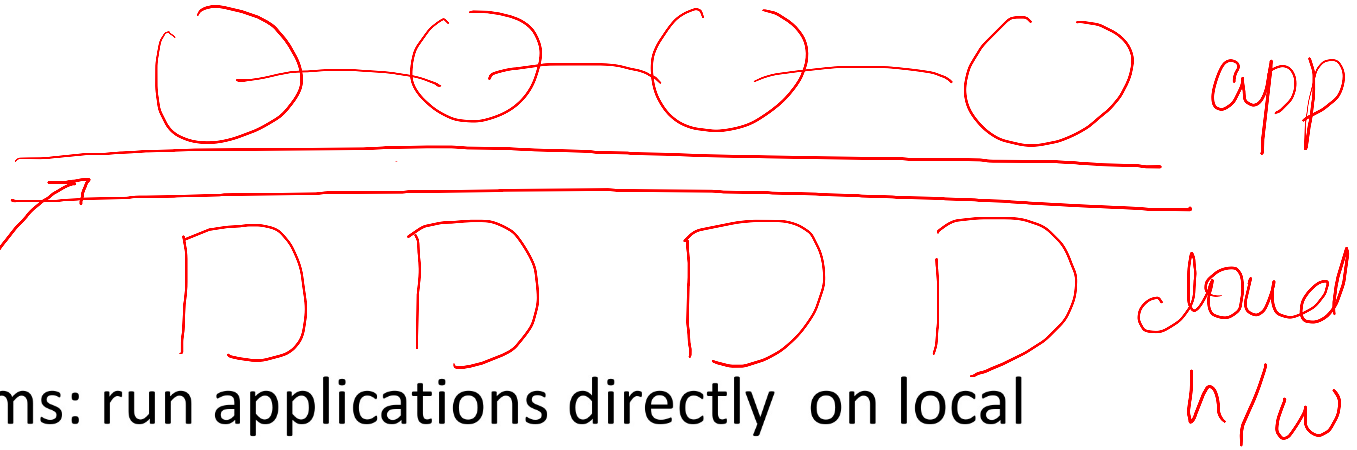
- The story so far: user programs run over OS, which runs on system hardware (CPU, main memory, I/O devices)
- Sometimes, we want to virtualize the entire system: make one system appear like multiple separate systems
  - Separate users, processes, operating systems, isolated from each other
- Why? Efficient sharing of hardware, better isolation than with processes
- How? Run multiple virtual machines (VMs) on the same underlying physical machine (PM)
  - Each VM runs its own separate guest OS, guest applications
- Another technique: containers (lightweight VMs)
- Renewed interest in virtualization due to popularity of cloud computing

What is cloud computing?



- Cloud: commodity servers with lots of compute (CPUs) and storage (memory, disk), connected with high speed networking, located in data centers
  - Usually setup by public cloud providers (Amazon, Azure, Google Cloud etc.) for access by anyone on demand, for a payment
- Many ways of interfacing with the cloud
  - Cloud providers manage infrastructure. Users access cloud infrastructure (e.g., a VM, storage space) and run their own applications on it (Infrastructure-as-a-service/IaaS)
  - Cloud providers setup software platforms, expose APIs. Users build cloud applications using platform APIs (Platform-as-a-Service/PaaS)
  - Cloud providers setup and manage complete software applications. Users access cloud software directly (Software-as-a-service/SaaS)
- Multiple users (tenants) share cloud servers using virtualization
  - Each tenant/user is given VMs/containers on a cloud server

Cloud applications

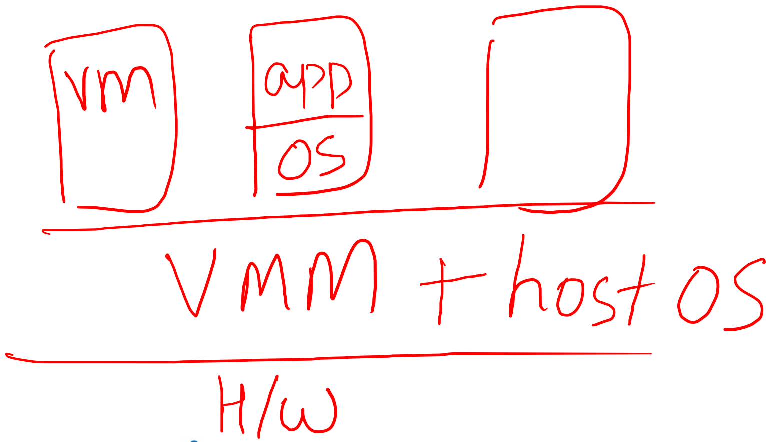


- Traditional way of building systems: run applications directly on local servers ("baremetal")
- Alternatives: run applications on a private cloud (within organization) or on public cloud (managed by cloud providers)
  - Multiple components of a computer system (front-end, back-end, web server, database, ..) run on separate VMs or containers on cloud infrastructure
- Cloud management and orchestration software eases management of VMs/containers on the cloud
  - Lifecycle management of VMs: creation, deletion, restart after crash
  - Placing VMs optimally on physical machines that are free
  - Migrating VMs across physical machines, e.g., in case of server maintenance
  - Instantiating additional replicas of components when under load (auto-scaling)
  - Examples: Openstack, Kubernetes

Pros and cons of cloud computing

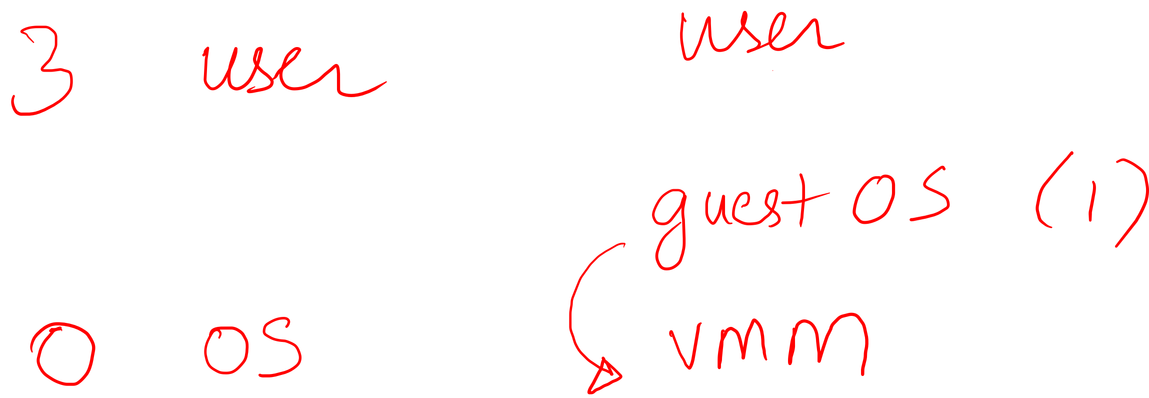
- Advantages of running applications on the cloud
  - Multiplexing gains: multiple VMs/containers can share hardware resources better
  - Orchestration: cloud orchestration simplifies running large systems
  - Hassle-free maintenance: hardware/software maintained by cloud providers
  - Pay as per usage: no need to invest in servers if only lightly used
  - Quick provisioning on demand: servers available immediately when needed
- Disadvantages of running applications on cloud
  - Worse performance: longer delay to access servers via internet
  - Higher cost: if cloud servers used heavily, maybe cheaper to have own servers

Virtual Machine Monitor (VMM)



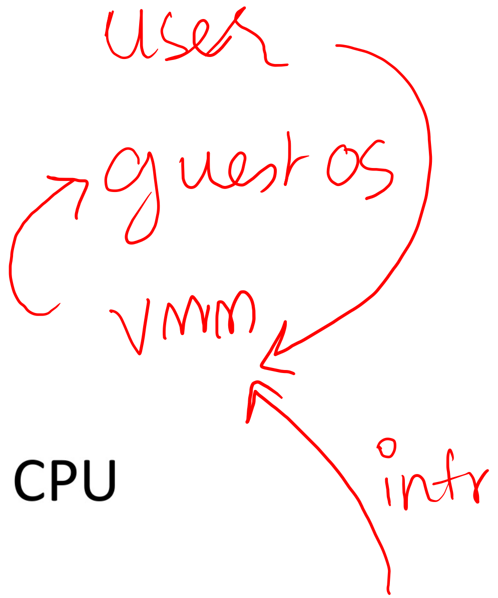
- How are VMs implemented?
- VMs run over a virtual machine monitor (VMM) or hypervisor
  - Type 1 hypervisor: VMM runs directly on hardware, includes OS functionality
  - Type 2 (hosted) hypervisor: VMM runs alongside existing host OS
  - The OS running inside VM is called guest OS
- VMM multiplexes VMs much like how OS multiplexes processes
  - VMM performs machine switch (much like context switch)
  - Run a VM for a bit, save context and switch to another VM, and so on…
- What is the challenge?
  - Guest OS expects to have unrestricted access to hardware, unlike user programs
  - But guest OS cannot be permitted privileged operations for security reasons

Trap and emulate VMM



- How to implement a VMM?
- All CPUs have multiple privilege levels
  - Ring 0,1,2,3 in x86 CPUs
  - Normally, user process in ring 3, OS in ring 0
  - Privileged instructions only run by OS in ring 0, not by user code
- Trap-and-emulate VMM: simple VMM design
  - Guest app in ring 3, guest OS in ring 1, VMM/host OS in ring 0
  - Guest OS is protected from guest apps, but not as privileged as VMM
  - Privileged operations of guest OS trap to VMM, VMM emulates action on behalf of guest
- Assumption: any instruction that accesses hardware is privileged, guest OS cannot execute instructions that access hardware without trapping to VMM

Trap and emulate VMM: examples



- Guest VM sets IDT (Interrupt Descriptor Table)
  - Setting IDT is privileged operation, traps to VMM
  - VMM remembers IDT of guest, but does not use guest IDT on CPU
  - VMM uses its own IDT that invokes VMM code on traps
- Guest user application makes system call
  - Traps to VMM, VMM jumps to guest OS code, system call handled by guest OS
- Guest OS initiates I/O operation by writing to I/O device register
  - Privileged instruction, traps to VMM, VMM performs the action for guest
- Interrupt arrives from I/O device
  - Traps to VMM, VMM finds the guest VM to which this interrupt belongs
  - VMM injects interrupt to guest, invokes the interrupt handler of guest VM

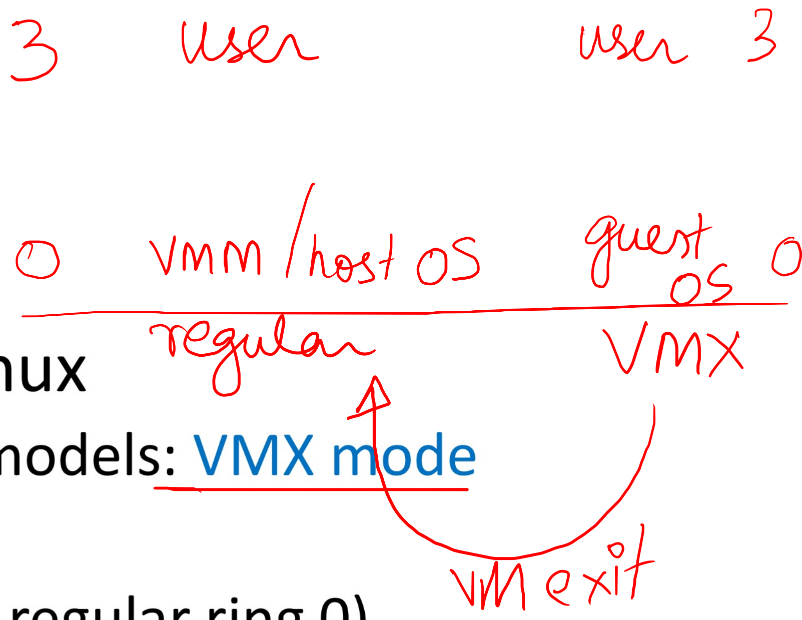
Problems with trap and emulate in x86

- Guest OS may realize it is running at lower privilege level
  - Some CPU registers indicate CPU privilege level
  - Guest OS can read these values and get offended!
- Some x86 instructions which change hardware state run in both privileged and unprivileged modes
  - Will behave differently when guest OS is in ring 0 vs in less privileged ring 1
  - OS behaves incorrectly in ring1, but will not trap to VMM
- Why these problems?
  - OS code not normally designed to run at a lower privilege level
  - Instruction set architecture of x86 not developed with virtualization in mind
- Simple trap and emulate idea does not work with x86 CPUs

Techniques to virtualize x86 (1)

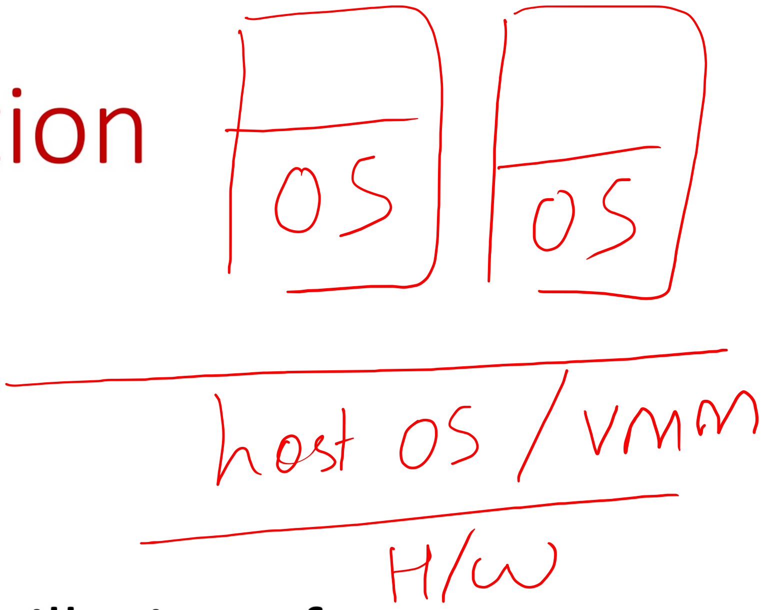
- Paravirtualization: rewrite guest OS code to be virtualizable
  - Guest OS won't invoke privileged operations, makes "hypercalls" to VMM
  - Needs OS source code changes, cannot work with unmodified OS
  - Example: Xen hypervisor
- Full virtualization: CPU instructions of guest OS binary/executable are translated to be virtualizable
  - Problematic instructions (e.g., access hardware but do not trap to VMM) are translated to trap to VMM
  - Translation of OS binary only, works with unmodified OS binary
  - Higher overhead than paravirtualization
  - Example: VMWare workstation

Techniques to virtualize x86 (2)

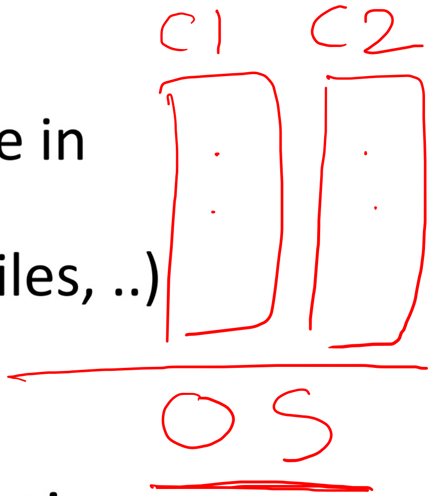


- Hardware assisted virtualization: KVM/QEMU in Linux
  - x86 CPU has added support for virtualization in recent models: VMX mode
  - 4 rings in regular mode, 4 rings in VMX mode
  - Guest OS is run in VMX mode ring 0 (not as powerful as regular ring 0)
  - VMM and host OS run in regular ring 0
  - VMM sets triggers (e.g., specific instructions, interrupts) which can cause VM exit from VMX mode to VMM/host OS in regular mode
  - No need to rewrite OS code, or translate OS binary
  - Best of both words: unmodified guest OS running in ring 0, VMM retains control on guest OS execution
  - Optimizations around reducing overhead of VM exits to improve performance

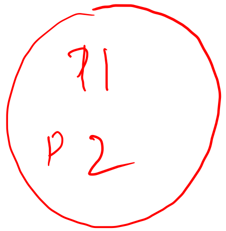
Containers: lightweight virtualization

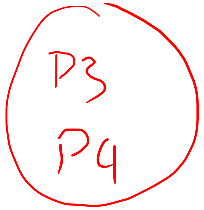


- Running multiple VMs imposes some overhead
  - Multiple guest OS images consume memory
  - Switching across guest VMs is expensive
- Containers: lightweight virtualization
- Containers share same base OS image, but provide illusion of different systems by having:
  - Different process trees (processes in one container cannot "see" those in other containers)
  - Different root file systems (libraries, system programs, configuration files, ..)
  - Resource usage limits enforced across containers
  - Other such mechanisms for isolation
- Containers have lesser overhead than VMs, but also lesser isolation

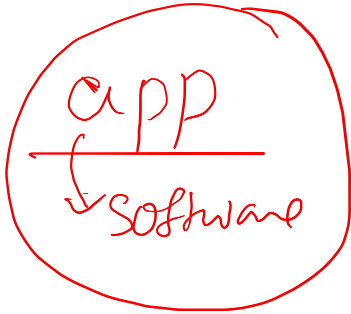


Container implementation

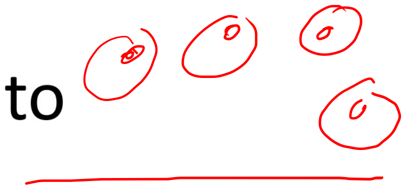




- Two mechanisms in Linux kernel over which containers are built:
  - Namespaces: a way to provide isolated view of a certain global resource (e.g., root file system or process tree) to a set of processes
  - Cgroups: a way to set resource limits on a group of processes
- Frameworks like LXC, Docker use these mechanisms to implement containers
  - LXC is general container framework, provides VM-like interface
  - Docker containers are optimized to run a single program (easy way to package an application and all its dependencies)



- Docker Swarm, Kubernetes: container orchestration frameworks to manage multiple containers across multiple physical machines
  - Kubernetes manages multiple "pods" of containers across multiple physical nodes
  - Lifecycle management of containers, autoscaling to handle overload, and so on.



Summary

- In this lecture:
  - Virtual machines, VMM / hypervisor
  - Techniques for virtualization
  - Containers
  - Cloud computing
- Try the following: setup a VM with a guest OS different from your host OS on your computer. Which virtualization technique does your VM use?
- Try the following: setup a container on your system. Observe the isolation that containers provide.