Virtual Machines

DM510 Operating Systems

Lars Rohwedder

Disclaimer

These slides contain (modified) content and media from the official Operating System Concepts slides:
https://www.os-book.com/OS10/slide-dir/index.html

Today's lecture

- Chapters 18 of course book
- Additional material on containers, specifically Docker

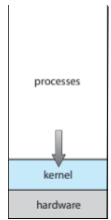
Virtual machine

Instead of directly running operating system on hardware, run it inside a virtual environment. Advantages:

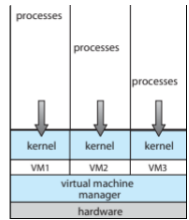
- Can run several operating systems **simultaneously** on one computer (better resource utilization in **cloud computing**)
- Isolation between users running in different VMs
- Security when testing operating systems
- Can run software in specific environment (for example, for compatibility)

VM Components

- **Host:** hardware of system
- **Virtual machine manager (VMM)** or **hypervisor**: responsible for creating and running VM environment
- **Guest:** operating system running inside VM



Traditional computer

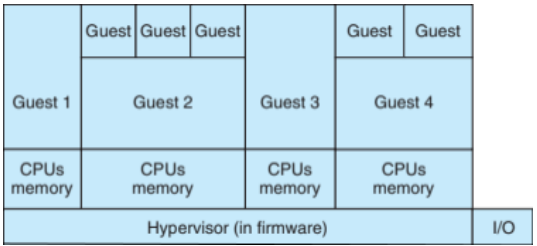


Computer with VMs

Types of Virtual Machines

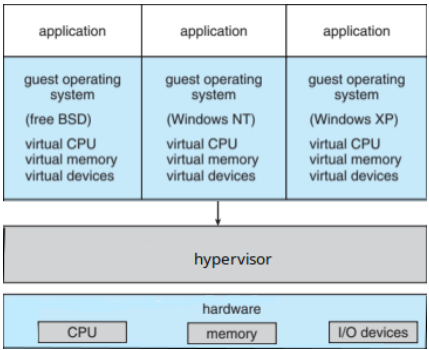
Type 0 hypervisors

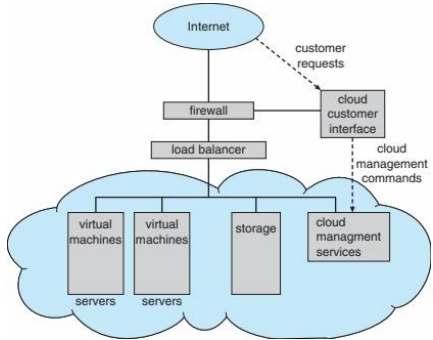
- Specialized hardware for the purpose of running several operating systems
- Hypervisor loaded directly from firmware (software shipped with the system)
- Each guest can have dedicated CPUs, memory, sometimes I/O devices
- Shared I/O devices may be managed by special guest, the control partition
- Requires specialized hardware. Tends to have fewer features due to high development costs



Type 1 hypervisors

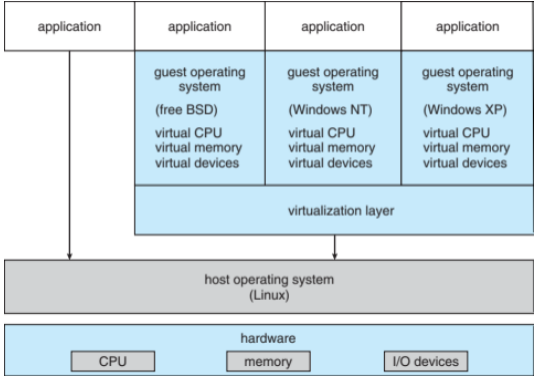
- VMM is special operating system running on standard hardware
- Very common in cloud computing
- Example: VMWare vSphere





Type 2 hypervisors

- VMM is user process running on standard operating system
- Most common for personal use
- Example: Oracle VirtualBox, VMWare Workstation



Other related technologies

Emulator

- Runs software on different processor architecture than it was compiled for
- Original processor architecture has to be simulated and each instruction needs to be interpreted by software ⇝ orders of magnitude slower execution

Other related technologies

Emulator

- Runs software on different processor architecture than it was compiled for
- Original processor architecture has to be simulated and each instruction needs to be interpreted by software ⇝ orders of magnitude slower execution

Programming environment virtualization

- Some higher programming languages are interpreted by software that virtualizes part of the environment
- Example: Java virtual machine
- Purpose of these environments is to run single application, not entire operating system, and to provide a level of abstraction for higher compatibility between environments
- Very different role from hypervisors

Other related technologies

Emulator

- Runs software on different processor architecture than it was compiled for
- Original processor architecture has to be simulated and each instruction needs to be interpreted by software ⇝ orders of magnitude slower execution

Programming environment virtualization

- Some higher programming languages are interpreted by software that virtualizes part of the environment
- Example: Java virtual machine
- Purpose of these environments is to run single application, not entire operating system, and to provide a level of abstraction for higher compatibility between environments
- Very different role from hypervisors

Containers

See second half of lecture

Other related technologies

Emulator

- Runs software on different processor architecture than it was compiled for
- Original processor architecture has to be simulated and each instruction needs to be interpreted by software $\rightsquigarrow$ orders of magnitude slower execution

Programming environment virtualization

- Some higher programming languages are interpreted by software that virtualizes part of the environment
- Example: Java virtual machine
- Purpose of these environments is to run single application, not entire operating system, and to provide a level of abstraction for higher compatibility between environments
- Very different role from hypervisors

Containers

See second half of lecture

Paravirtualization

- Guest OS is aware of VM environment
- It is specially compiled to run in VM and cooporates with host

Virtual Machine Implementation

CPU and memory

- VMM maintains data structure for state of CPU(s) and memory for VMs
- vCPU (virtual CPU) can be switched on and off the actual hardware (implementation differs, see next slides)
- Similarly, page tables for each VM are maintained and switched by VMM

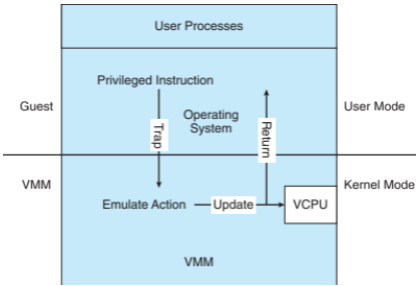
Generally, these methods cause overhead, more TLB missed, and generally worse performance

Trap and emulate (vCPU implementation)

We can execute unprivileged instructions of guest normally on CPU, but we cannot allow VM to switch to kernel mode, since this would allow direct hardware access for the guest

Trap and emulate

- The guest OS (including the guest kernel) is executed as a user process
- Recall: When a user process executes a priviledged instruction, the CPU generates an interrupt (trap) and normally the kernel's interrupt handler would terminate the process
- In trap-and-emulate, the VMM is registered as interrupt handler and **emulates** the priviledged instructions (simulates their effect). The process (guest OS) is not terminated
- Emulation is much slower, but it can be improved by hardware support, especially additional modes (besides user and kernel)

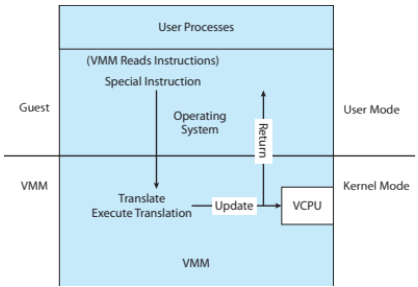


Binary translation

Problem with trap and emulate: on some CPUs (e.g. Intel x86), some “special” instructions behave differently in user or kernel mode and do not generate trap if in user mode

Binary translation

- When vCPU is in kernel mode (or rather: it believes to be) VMM examines next instructions and replaces special ones by equivalent non-special instructions on-the-fly
- Various optimizations including code caching (to translate only once) make it practical. For example, VMWare reports only 5% slowdown in a test

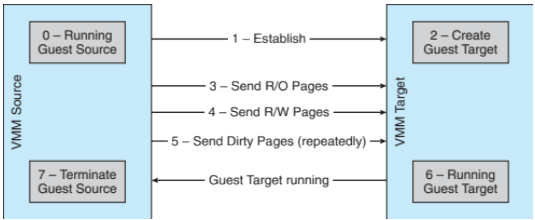


Snapshots and live migration

- VMM can create **snapshots** containing all information about current state of VM

Live migration

- Snapshots can also be used to migrate VM from one host to another
- Optimized variant: send snapshot to target host, while source is still running keep sending updates on changed (dirty) memory pages, once update cycle small enough suspend source, send last dirty pages and launch target
- No downtime. Can even keep connections (TCP, etc.) up



Containers

Motivation for containers

Application requirements can be complicated

- programming language interpreters of specific versions
- runtime libraries of specific versions
- where to find required files in file system

Makes it more difficult to

- migrate application to another system
- replicate scientific experiments
- test application under same environment as it will be deployed

Can be solved by running application in virtual machine (with fixed environment), but running entire operating system for specific application may not always be necessary

Containers

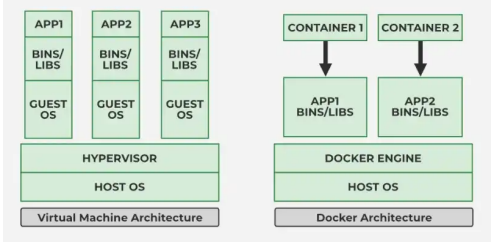
Linux kernel provides features that allows to configure per-process environment

- **namespaces**: isolate specific processes. For example, to restrict which other devices, processes, files, etc. it can see
- **cgroups**: restrict resource usage (CPU, memory, etc.) of specific processes

Used by container software (for example **Docker**) to isolate programs and run them in specific environment

Containers vs. virtual machines

- Container environments are less flexible, since they do not have their own kernel. For example, Windows application cannot be run inside Linux
- Containers are light-weight and therefore potentially more efficient
- Containerization helps with security and protection, since applications are run inside **sandbox** (should not affect anything outside its container), but less secure than virtual machines (for example, in case of OS bugs)



Source: `www.geeksforgeeks.org/devops/introduction-to-docker/`

Docker

- Very popular container software
- Runs on Linux (or Linux inside VM)
- Uses kernel functions (namespaces, cgroups) to implement containerization
- Software requirements (for example, Python packages) can be specified easily to automatically create image for application environment created

Docker image

- Docker image contains all data (libraries, application code, etc.) of a docker container to run it
- Multiple instances of image can be run
- Images can have parent images that they inherit from. For example, python applications usually use the “python” image as parent
- **Dockerfile** specifies what should be built into image and what commands should be executed on launch