IMPLEMENTATION LEVELS OF VIRTUALIZATION

Virtualization is a computer architecture technology by which multiple virtual machines (VMs) are multiplexed in the same hardware machine. The purpose of a VM is to enhance resource sharing by many users and improve computer performance in terms of resource utilization and application flexibility.

Hardware resources (CPU, memory, I/O devices, etc.) or software resources (operating system and software libraries) can be virtualized in various functional layers.

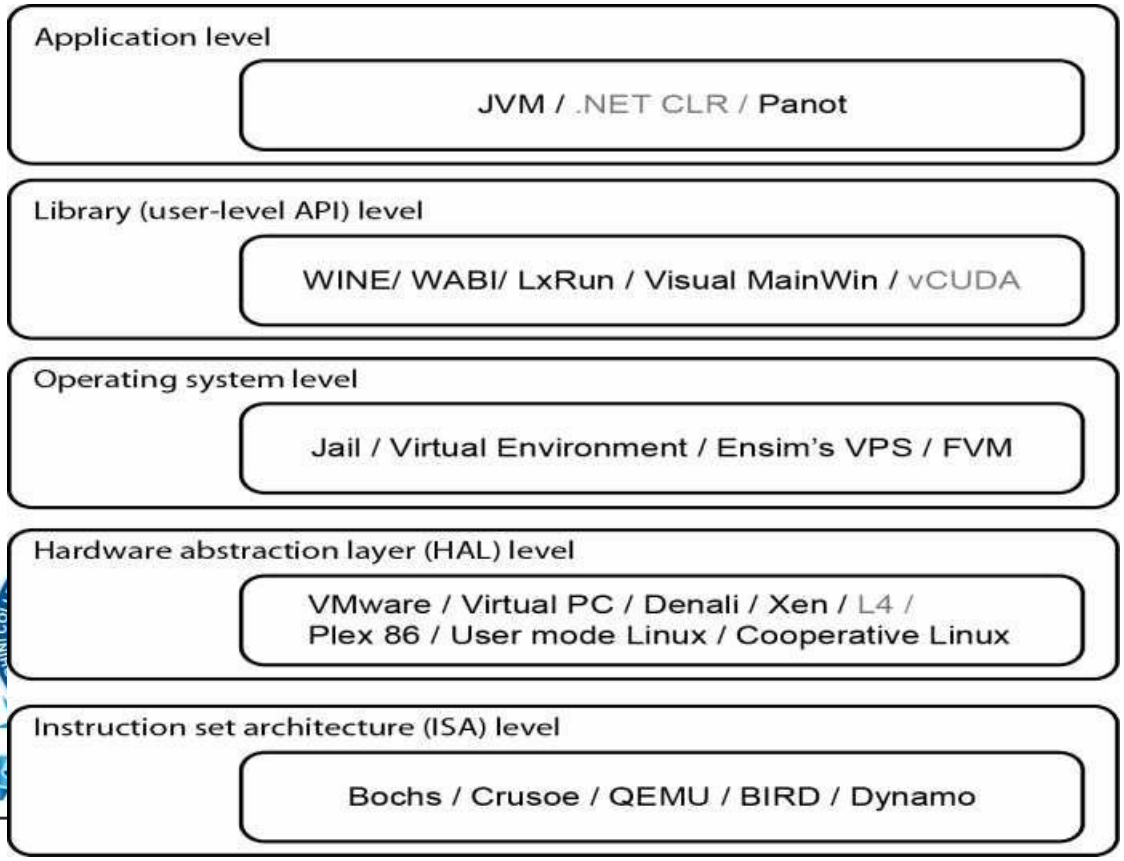
The idea is to separate the hardware from the software to yield better system efficiency. For example, computer users gained access to much enlarged memory space when the concept of virtual memory was introduced. Similarly, virtualization techniques can be applied to enhance the use of compute engines, networks and storage.

Levels of Virtualization:

A traditional computer runs with host operating system specially tailored for its hardware architecture, as shown in Figure 2.11 (a). After virtualization, different user applications managed by their own operating systems (guest OS) can run on the same hardware, independent of the host OS.

This is often done by adding additional software, called a virtualization layer as shown in Figure 2.11 (b). This virtualization layer is known as hypervisor or virtual machine monitor (VMM). The VMs are shown in the upper boxes, where applications run with their own guest OS over the virtualized CPU, memory, and I/O resources. The main function of the software layer for virtualization is to virtualize the physical hardware of a host machine into virtual resources to be used by the VMs, exclusively. The virtualization software creates the abstractionofVMsbyinterposingavirtualizationlayeratvariouslevelsofacomputersystem.

Common virtualization layers include the instruction set architecture (ISA) level, hardware level, operating system level, library support level, and application level.



Virtualization ranging from hardware to applications in five abstraction levels.

Instruction Set Architecture Level:

At the ISA level, virtualization is performed by emulating a given ISA by the ISA of the host machine. For example, MIPS binary code can run on an x86 –based host machine with the help of ISA emulation. With this approach, it is possible to run a large amount of legacy binary code written for various processors on any given new hardware host machine. Instruction set emulation leads to virtual ISAs created on any hardware machine.

The basic emulation method is through code interpretation. An interpreter program interprets the source instructions to target instructions one by one. OneSource instruction may require tensorhundredsofnativetargetinstructionstoperformitsfunction.Obviously, this process is relatively slow. For better performance, dynamic binary translation is desired.

This approach translates basic blocks of dynamic source instructions to target instructions. The basic blocks can also be extended to program traces or super blocks to increase translation efficiency. Instruction set emulation requires binary translation and optimization. A virtual instruction set architecture (V-ISA) thus requires adding a processor-specific software translation layer to the compiler.

Hardware Abstraction Level:

Hardware-level virtualization is performed right on top of the bare hardware. The idea is to virtualize a computer's resources, such as its processors, memory, and I/O devices. The intention is to upgrade the hardware utilization rate by multiple users concurrently.

Operating System Level:

This refers to an abstraction layer between traditional OS and user applications. OS-level virtualization creates isolated containers on a single physical server and the OS instances to utilize the hardware and software in datacenters.

The containers behave like real servers. OS-level virtualization is commonly used in creating virtual hosting environments to allocate hardware resources among a large number of mutually distrusting users. It is also used, to a lesser extent, in consolidating server hardware by moving services on separate hosts into containers or VMs on one server.

Library Support Level:

Most applications use APIs exported by user level libraries rather than using lengthy system calls by the OS. Since most systems provide well documented APIs, such an interface becomes another candidate for virtualization.

Virtualization with library interfaces is possible by controlling the communication link between applications and the rest of a system through API hooks. The software tool WINE has implemented this approach to support Windows applications on top of UNIX hosts. Another example is the vCUDA which allows applications executing within VMs to leverage GPU hardware acceleration.

User-Application Level:

Virtualization at the application level virtualizes an application as a VM. On a traditional OS, an application often runs as a process. Therefore, application-level virtualization is also known as process-level virtualization. The most popular approach is to deploy high level language (HLL) VMs.

Relative Merits of Virtualization at Various Levels (More "X"'s Means Higher Merit, with a Maximum of 5 X's)

Level of Implementation	Higher Performance	Application Flexibility	Implementation Complexity	Application Isolation
ISA	X	XXXXX	XXX	XXX
Hardware-level virtualization	XXXXX	XXX	XXXXX	XXXX
OS-level virtualization	XXXXX	XX	XXX	XX
Runtime library support	XXX	XX	XX	XX
User application level	XX	XX	XXXXX	XXXXX

VMM Design Requirements and Providers

Hardware-level virtualization inserts a layer between real hardware and traditional operating systems. This layer is commonly called the Virtual Machine Monitor (VMM) and it manages the hardware resources of a computing system. Each time programs access the hardware the VMM captures the process. VMM acts as a traditional OS.

One hardware component, such as the CPU, can be virtualized as several virtual copies. Therefore, several traditional operating systems which are the same or different can sit on the same set of hardware simultaneously.

Three requirements for a VMM

• First, a VMM should provide an environment for programs which is essentially identical to the original machine.

• Second, programs run in this environment should show, at worst, only minor decreases in speed.

• Third, a VMM should be incomplete control of the system resources.

Provider and References	Host CPU	Host OS	Guest OS	Architecture
VMware Workstation [71]	x86, x86-64	Windows, Linux	Windows, Linux, Solaris, FreeBSD, Netware, OS/2, SCO, BeOS, Darwin	Full Virtualization
VMware ESX Server [71]	x86, x86-64	No host OS	The same as VMware Workstation	Para-Virtualization
Xen [7,13,42]	x86, x86-64, IA-64	NetBSD, Linux, Solaris	FreeBSD, NetBSD, Linux, Solaris, Windows XP and 2003 Server	Hypervisor
KVM [31]	x86, x86-64, IA-64, S390, PowerPC	Linux	Linux, Windows, FreeBSD, Solaris	Para-Virtualization

Virtualization Support at the OS Level

With the help of VM technology, a new computing mode known as cloud computing is emerging. Cloud computing is transforming the computing landscape by shifting the hardware and staffing costs of managing a computational center to third parties, just like banks. However, cloud computing has at least two challenges.

- The first is the ability to use a variable number of physical machines and VM instances depending on the needs of a problem.
- These cond challenge concerns the slow operation of instantiating newVMs.

Currently, new VMs originate either as fresh boots or as replicates of a template VM, unaware of the current application state. Therefore, to better support cloud computing, a large amount of research and development should be done.

Why OS-Level Virtualization?

To reduce the performance overhead of hardware-level virtualization, even hardware modification is needed. OS-level virtualization provides safe a sable solution for these hardware- level virtualization issues. Operating system virtualization inserts a virtualization

layer inside an operating system to partition a machine's physical resources. It enables multiple isolated VMs within a single operating system kernel. This kind of VM is often called a virtual execution environment (VE), Virtual Private System (VPS), or simply container. From the user's point of view, VEs look like real servers. This means aVE has its own set of processes, file system, user accounts, network interfaces with IP addresses, routing tables, firewall rules, and other personal settings. Although VEs can be customized for different people, they share the same operating system kernel.

Advantages of OS Extensions

(1) VMs at the operating system level have minimal startup/shutdown costs, low resource requirements, and high scalability.

(2) For an OS-level VM, it is possible for a VM and its host environment to synchronize state changes when necessary.

These benefits can be achieved via two mechanisms of OS-level virtualization:

(1) All OS-level VMs on the same physical machine share a single operating system kernel.

(2) The virtualization layer can be designed in a way that al lows processes in VMs to access as many resources of the host machine as possible, but never to modify them.

Virtualization on Linux or Windows Platforms

(1) Virtualization support on the Windows-based platform is still in the research stage. The Linux kernel offers an abstraction layer to allow software processes to work with and operate on resources without knowing the hardware details. New hardware may need a new Linux kernel to support. Therefore, different Linux platforms use patched kernels to provide special support for extended functionality.

Virtualization Support for Linux and Windows NT Platforms

Virtualization Support and Source of Information	Brief Introduction on Functionality and Application Platforms
Linux vServer for Linux platforms (http://linux-vserver.org/)	Extends Linux kernels to implement a security mechanism to help build VMs by setting resource limits and file attributes and changing the root environment for VM isolation
OpenVZ for Linux platforms [65]; http://ftp.openvz.org/doc/OpenVZ-Users-Guide.pdf)	Supports virtualization by creating *virtual private servers (VPSes)*; the VPS has its own files, users, process tree, and virtual devices, which can be isolated from other VPSes, and checkpointing and live migration are supported
FVM (Feather-Weight Virtual Machines) for virtualizing the Windows NT platforms [78])	Uses system call interfaces to create VMs at the NY kernel space; multiple VMs are supported by virtualized namespace and copy-on-write

Middle ware Support for Virtualization

Library-level virtualization is also known as user-level Application Binary Interface (ABI) or API emulation. This type of virtualization can create execution environments for running alien programs on a platform rather than creating a VM to run the entire operating system.API call interception and remapping are the key functions performed. This provides an overview of several library-level virtualization systems: namely the Windows Application Binary Interface (WABI), lxrun, WINE, Visual MainWin, and Vcuda.

Middleware and Library Support for Virtualization

Middleware or Runtime Library and References or Web Link	Brief Introduction and Application Platforms
WABI (http://docs.sun.com/app/docs/doc/802-6306)	Middleware that converts Windows system calls running on x86 PCs to Solaris system calls running on SPARC workstations
Lxrun (Linux Run) (http://www.ugcs.caltech.edu/~steven/lxrun/)	A system call emulator that enables Linux applications written for x86 hosts to run on UNIX systems such as the SCO OpenServer
WINE (http://www.winehq.org/)	A library support system for virtualizing x86 processors to run Windows applications under Linux, FreeBSD, and Solaris
Visual MainWin (http://www.mainsoft.com/)	A compiler support system to develop Windows applications using Visual Studio to run on Solaris, Linux, and AIX hosts
vCUDA (Example 3.2) (IEEE *IPDPS* 2009 [57])	Virtualization support for using general-purpose GPUs to run data-intensive applications under a special guest OS