UNIT V VIRTUAL MACHINES AND MOBILE OS

Virtual Machines - History, Benefits and Features, Building Blocks, Types of Virtual Machines and their Implementations, Virtualization and Operating-System Components; Mobile OS - iOS and Android.

Virtual Machines

Fundamental idea – abstract hardware of a single computer into several different execution environments

- Similar to layered approach
- But layer creates virtual system (virtual machine, or VM) on which operation systems or applications can run

Several components

- **Host** – underlying hardware system
- **Virtual machine manager (VMM) or hypervisor** – creates and runs virtual machines by providing interface that is identical to the host (Except in the case of paravirtualization)
- **Guest** – process provided with virtual copy of the host, Usually an operating system

Single physical machine can run multiple operating systems concurrently, each in its own virtual machine.

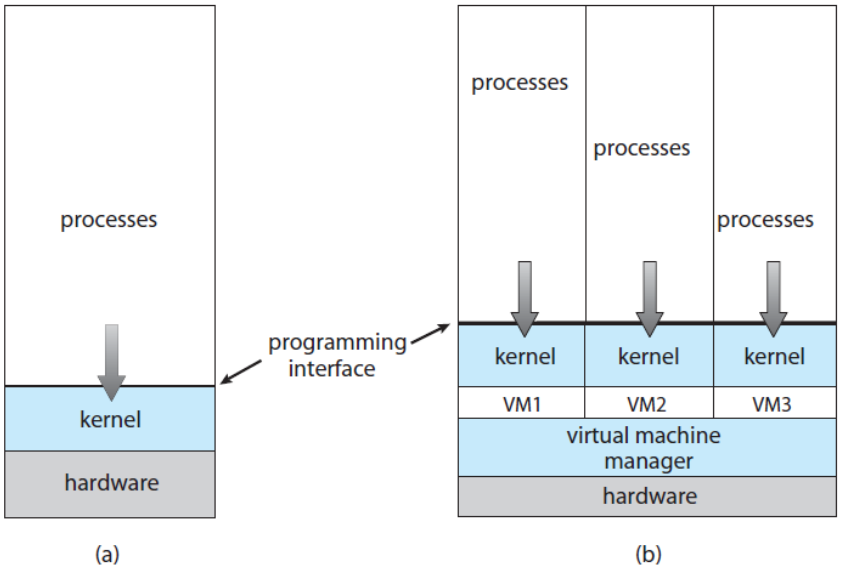


Figure 16.1 System models. (a) Nonvirtual machine. (b) Virtual machine.

The implementation of VMMs varies greatly. Options include the following:

• Hardware-based solutions that provide support for virtual machine creation and management via firmware. These VMMs, which are commonly found in mainframe and large to midsized servers, are generally known as **type 0 hypervisors**. IBM LPARs and Oracle LDOMs are examples.

• Operating-system-like software built to provide virtualization, including VMware ESX(mentioned above), Joyent SmartOS, and Citrix XenServer. These VMMs are known as **type 1 hypervisors.**

• General-purpose operating systems that provide standard functions as well as VMM functions, including MicrosoftWindows Server with HyperV and RedHat Linux with the KVM feature. Because such systems have a feature set similar to type 1 hypervisors, they are also known as **type 1.**

• Applications that run on standard operating systems but provide VMM features to guest operating systems. These applications, which include VMware Workstation and Fusion, Parallels Desktop, and Oracle Virtual-Box, are **type 2 hypervisors.**

• **Paravirtualization**, a technique in which the guest operating system is modified to work in cooperation with the VMM to optimize performance.

• **Programming-environment virtualization,** in which VMMs do not virtualize real hardware but instead create an optimized virtual system. This technique is used by Oracle Java and Microsoft.Net.

• **Emulators** that allow applications written for one hardware environment to run on a very different hardware environment, such as a different type of CPU.

• **Application containment,** which is not virtualization at all but rather provides virtualization-like features by segregating applications from the operating system. Oracle Solaris Zones, BSD Jails, and IBM AIX WPARs "contain" applications, making them more secure and manageable.

History

Virtual machines first appeared commercially on IBM mainframes in 1972.

Virtualization was provided by the IBM VM operating system. This system has evolved and is still available.

IBM VM370 divided a mainframe into multiple virtual machines, each running its own operating system.

A major difficulty with the VM approach involved disk systems. Suppose that the physical machine had three disk drives but wanted to support seven virtual machines. Clearly, it could not allocate a disk drive to each virtual machine.

The solution was to provide virtual disks-termed minidisks in IBM's VM operating system. The minidisks are identical to the system's hard disks in all respects except size. The system implemented each minidisk by allocating as many tracks on the physical disks as the minidisk needed.

Once the virtual machines were created, users could run any of the operating systems or software packages that were available on the underlying machine.

For the IBM VM system, a user normally ran CMS a single-user interactive operating system.

For many years after IBM introduced this technology, a formal definition of virtualization helped to establish system requirements and a target for functionality.

The virtualization requirements stated that:

1. AVMM provides an environment for programs that is essentially identical to the original machine.
2. Programs running within that environment show only minor performance decreases.
3. The VMM is in complete control of system resources.

These requirements of fidelity, performance, and safety still guide virtualization efforts today.

By the late 1990s, Intel 80x86 CPUs had become common, fast, and rich in features.

Accordingly, developers launched multiple efforts to implement virtualization on that platform.

Both Xen and VMware created technologies, still used today, to allow guest operating systems to run on the 80x86.

Since that time, virtualization has expanded to include all common CPUs, many commercial and open-source tools, and many operating systems.

For example, the open-source VirtualBox project (http://www.virtualbox.org) provides a program than runs on Intel x86 and AMD64 CPUs and on Windows, Linux, Mac OS X, and Solaris host operating systems.

Possible guest operating systems include many versions of Windows, Linux, Solaris, and BSD, including even MS-DOS and IBM OS/2.

Benefits and Features

One important advantage of virtualization is **that the host system is protected from the virtual machines**, just as the virtual machines are protected from each other.

A virus inside a guest operating system might damage that operating system but is unlikely to affect the host or the other guests. Because each virtual machine is almost completely isolated from all other virtual machines, there are almost no protection problems.

Two approaches to provide sharing have been implemented. First, it is possible to share a **file-system volume** and thus to share files. Second, it is possible to define a **network of virtual machines**, each of which can send information over the virtual communications network.

The network is modelled after physical communication networks but is implemented in software. Of course, the VMM is free to allow any number of its guests to use physical resources, such as a physical network connection (with sharing provided by the VMM), in which case the allowed guests could communicate with each other via the physical network.

One feature common to most virtualization implementations is the ability to **freeze, or suspend,** a running virtual machine.

But VMMs go one step further and **allow copies and snapshots** to be made of the guest.

The copy can be used to create a new VM or to move a VM from one machine to another with its current state intact.

The guest can then resume where it was, as if on its original machine, creating a clone.

The snapshot records a point in time, and the guest can be reset to that point if necessary (for example, if a change was made but is no longer wanted).

Often, VMMs allow many snapshots to be taken. For example, snapshots might record a guest's state every day for a month, making restoration to any of those snapshot states possible. These abilities are used to good advantage in virtual environments.

A virtual machine system is a perfect vehicle for operating-system research and development.

Furthermore, the operating system runs on and controls the entire machine, meaning that the system must be stopped and taken out of use while changes are made and tested. This period is commonly called system-development time. Since it makes the system unavailable to users, system-development time on shared systems is often scheduled late at night or on weekends, when system load is low.

A virtual-machine system can eliminate much of this latter problem. System programmers are given their own virtual machine, and system development is done on the virtual machine instead of on a physical machine.

Normal system operation is disrupted only when a completed and tested change is ready to be put into production. Another advantage of virtual machines for developers is that multiple operating systems can run concurrently on the developer's workstation.

This virtualized workstation allows for rapid porting and testing of programs in varying environments. In addition, multiple versions of a program can run, each in its own isolated operating system, within one system.

A major advantage of virtual machines in production **data-center** use is system consolidation, which involves taking two or more separate systems and running them in virtual machines on one system. Such physical-to-virtual conversions result in resource optimization, since many lightly used systems can be combined to create one more heavily used system.

Consider, too, that management tools that are part of the VMM allow system administrators to manage many more systems than they otherwise could. A virtual environment might include 100 physical servers, each running 20 virtual servers. Without virtualization, 2,000 servers would require several system administrators. With virtualization and its tools, the same work can be managed by one or two administrators.

One of the tools that make this possible is **templating**, in which one standard virtual machine image, including an installed and configured guest operating system and applications, is saved and used as a source for multiple running VMs.

Other features include managing the patching of all guests, **backing up** and **restoring** the guests, and monitoring their resource use. Virtualization can improve not only resource utilization but also resource management.

Some VMMs include a **live migration** feature that moves a running guest from one physical server to another without interrupting its operation or active network connections.

If a server is overloaded, live migration can thus free resources on the source host while not disrupting the guest.

Similarly, when host hardware must be repaired or upgraded, guests can be migrated to other servers, the evacuated host can be maintained, and then the guests can be migrated back.

This operation occurs without downtime and without interruption to users.

Building Blocks

Although the virtual machine concept is useful, it is difficult to implement. Much work is required to provide an exact duplicate of the underlying machine. This is especially a challenge on dual-mode systems, where the underlying machine has only user mode and kernel mode. In this section, we examine the building blocks that are needed for efficient virtualization.

VMMs use several techniques to implement virtualization, including trap-and-emulate and binary translation.

One important concept found in most virtualization options is the implementation of a virtual CPU (VCPU).

The VCPU does not execute code. Rather, it represents the state of the CPU as the guest machine believes it to be.

For each guest, the VMM maintains a VCPU representing that guest's current CPU state. When the guest is context-switched onto a CPU by the VMM, information from the VCPU is used to load the right context, much as a general-purpose operating system would use the PCB.

Trap-and-Emulate

On a typical dual-mode system, the virtual machine guest can execute only in user mode (unless extra hardware support is provided).

The kernel, of course, runs in kernel mode, and it is not safe to allow user-level code to run in kernel mode. Just as the physical machine has two modes, however, so must the virtual machine.

Consequently, we must have a virtual user mode and a virtual kernel mode, both of which run in physical user mode. Those actions that cause a transfer from user mode to kernel mode on a real machine (such as a

system call, an interrupt, or an attempt to execute a privileged instruction) must also cause a transfer from virtual user mode to virtual kernel mode in the virtual machine.

How can such a transfer be accomplished? The procedure is as follows: When the kernel in the guest attempts to execute a privileged instruction, that is an error (because the system is in user mode) and causes a trap to the VMM in the real machine.

The VMM gains control and executes (or "emulates") the action that was attempted by the guest kernel on the part of the guest. It then returns control to the virtual machine. This is called the trap-and-emulate method and is shown in Figure 16.2. Most virtualization products use this method to one extent or other.

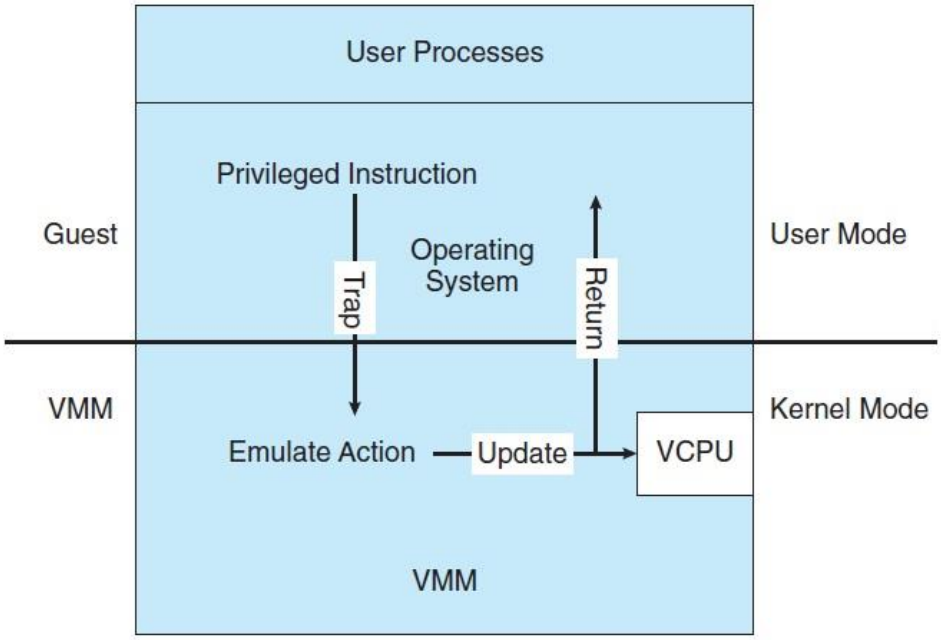


Figure 16.2 Trap-and-emulate virtualization implementation.

With privileged instructions, time becomes an issue. All non-privileged instructions run natively on the hardware, providing the same performance for guests as native applications. Privileged instructions create extra overhead, however, causing the guest to run more slowly than it would natively. In addition, the CPU is being multiprogrammed among many virtual machines, which can further slowdown the virtual machines in unpredictable ways.

Binary Translation

Some CPUs do not have a clean separation of privileged and non-privileged instructions.

Let's consider an example of the problem.

The command popf loads the flag register from the contents of the stack.

If the CPU is in privileged mode, all of the flags are replaced from the stack.

If the CPU is in user mode, then only some flags are replaced, and others are ignored.

Because no trap is generated

if popf is executed in user mode, the trap-and-emulate procedure is rendered useless.

Binary translation is fairly simple in concept but complex in implementation. The basic steps are as follows:
1. If the guest VCPU is in user mode, the guest can run its instructions natively on a physical CPU.
2. If the guest VCPU is in kernel mode, then the guest believes that it is running in kernel mode.
The VMM examines every instruction the guest executes in virtual kernel mode by reading the next few instructions that the guest is going to execute, based on the guest's program counter.
Instructions other than special instructions are run natively.
Special instructions are translated into a new set of instructions that perform the equivalent task, for example changing the flags in the VCPU.

Binary translation is shown in Figure 16.3. It is implemented by translation code within the VMM.
The code reads native binary instructions dynamically from the guest, on demand, and generates native binary code that executes in place of the original code.
The basic method of binary translation just described would execute correctly but perform poorly. Fortunately, the vast majority of instructions would execute natively.
But how could performance be improved for the other instructions? We can turn to a specific implementation of binary translation, the VMware method, to see one way of improving performance. Here, caching be translated is cached. All later executions of that instruction run from the translation cache and need not be translated again. If the cache is large enough, this method can greatly improve performance.

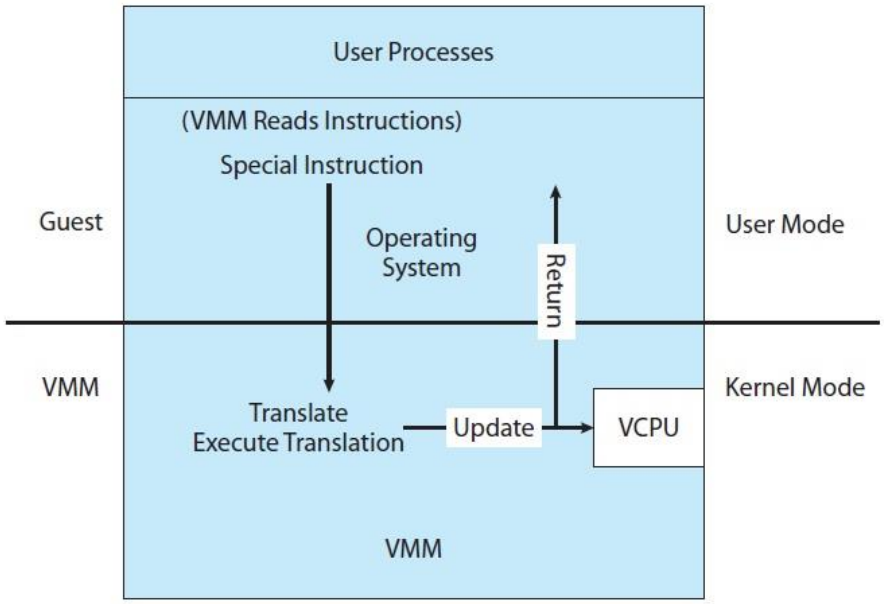


Figure 16.3 Binary translation virtualization implementation.

Types of Virtual Machines and Their Implementations

Type 0 Hypervisor

Type 0 hypervisors have existed for many years under many names, including "partitions" and "domains". They are a hardware feature, and that brings its own positives and negatives.

The VMM itself is encoded in the firmware and loaded at boot time. In turn, it loads the guest images to run in each partition. The feature set of a type 0 hypervisor tends to be smaller than those of the other types because it is implemented in hardware.

In the control partition, a guest operating system provides services (such as networking) via daemons to other guests, and the hypervisor routes I/O requests appropriately.

Some type 0 hypervisors are even more sophisticated and can move physical CPUs and memory between running guests.

In these cases, the guests are paravirtualized, aware of the virtualization and assisting in its execution. For example, a guest must watch for signals from the hardware or VMM that a hardware change has occurred, probe its hardware devices to detect the change, and add or subtract CPUs or memory from its available resources.

Because type 0 virtualization is very close to raw hardware execution, it should be considered separately from the other methods discussed here.

A type 0 hypervisor can run multiple guest operating systems (one in each hardware partition).

All of those guests, because they are running on raw hardware, can in turn be VMMs. Essentially, the guest operating systems in a type 0 hypervisor are native operating systems with a subset of hardware made available to them. Because of that, each can have its own guest operating systems (Figure 16.5).

Other types of hypervisors usually cannot provide this virtualization-within-virtualization functionality.

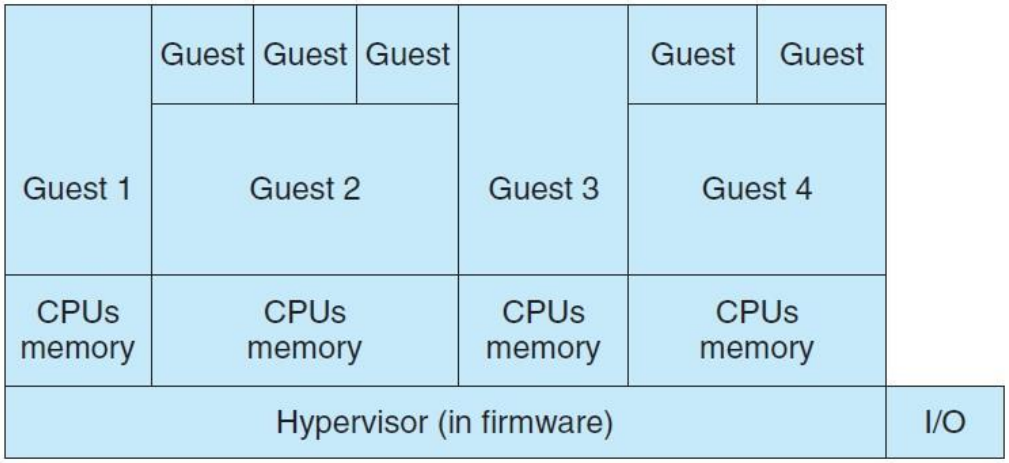


Figure 16.5 Type 0 hypervisor.

Type 1 Hypervisor

Type 1 hypervisors are commonly found in company data centers and are in a sense becoming "the data-center operating system."

They are special-purpose operating systems that run natively on the hardware, but rather than providing system calls and other interfaces for running programs, they create, run, and manage guest operating systems.

In addition to running on standard hardware, they can run on type 0 hypervisors, but not on other type 1 hypervisors. Whatever the platform, guests generally do not know they are running on anything but the native hardware.

Type 1 hypervisors run in kernel mode, taking advantage of hardware protection.

Where the host CPU allows, they use multiple modes to give guest operating systems their own control and improved performance.

They implement device drivers for the hardware they run on, because no other component could do so. Because they are operating systems, they must also provide CPU scheduling, memory management, I/O management, protection, and even security.

Frequently, they provide APIs, but those APIs support applications in guests or external applications that supply features like backups, monitoring, and security. Many type 1 hypervisors are closed-source commercial offerings, such as VMware ESX while some are open source or hybrids of open and closed source, such as Citrix XenServer and its open Xen counterpart.

By using type 1 hypervisors, data-center managers can control and manage the operating systems and applications in new and sophisticated ways. An important benefit is the ability to consolidate more operating systems and applications onto fewer systems.

In many ways, they treat a guest operating system as just another process, able it with special handling provided when the guest tries to execute special instructions.

Type 2 Hypervisor

Type 2 hypervisors are less interesting to us as operating-system explorers, because there is very little operating-system involvement in these application level virtual machine managers. This type of VMM is simply another process run and managed by the host, and even the host does not know virtualization is happening within the VMM.

Type 2 hypervisors have limits not associated with some of the other types. For example, a user needs administrative privileges to access many of the hardware assistance features of modern CPUs. If the VMM is being run by a standard user without additional privileges, the VMM cannot take advantage of these features. Due to this limitation, as well as the extra overhead of running a general-purpose operating system as well as guest operating systems, type 2 hypervisors tend to have poorer overall performance than type 0 or 1.

As is often the case, the limitations of type 2 hypervisors also provide some benefits. They run on a variety of general-purpose operating systems, and running them requires no changes to the host operating system. A student can use a type 2 hypervisor, for example, to test a non-native operating system without replacing the native operating system. In fact, on an Apple laptop, a student could have versions of Windows, Linux, Unix, and less common operating systems all available for learning and experimentation.

Paravirtualization

As we've seen, paravirtualization takes a different tack than the other types of virtualization.

Rather than try to trick a guest operating system into believing it has a system to itself, paravirtualization presents the guest with a system that is similar but not identical to the guest's preferred system.

The guest must be modified to run on the paravirtualized virtual hardware.

The gain for this extra work is more efficient use of resources and a smaller virtualization layer.

The Xen VMM, which is the leader in paravirtualization, has implemented several techniques to optimize the performance of guests as well as of the host system.
For example, as we have seen, some VMMs present virtual devices to guests that appear to be real devices. Instead of taking that approach, the Xen VMM presents clean and simple device abstractions that allow efficient I/O, as well as good communication between the guest and the VMM about device I/O.
For each device used by each guest, there is a circular buffer shared by the guest and the VMM via shared memory. Read and write data are placed in this buffer, as shown in Figure 16.6.

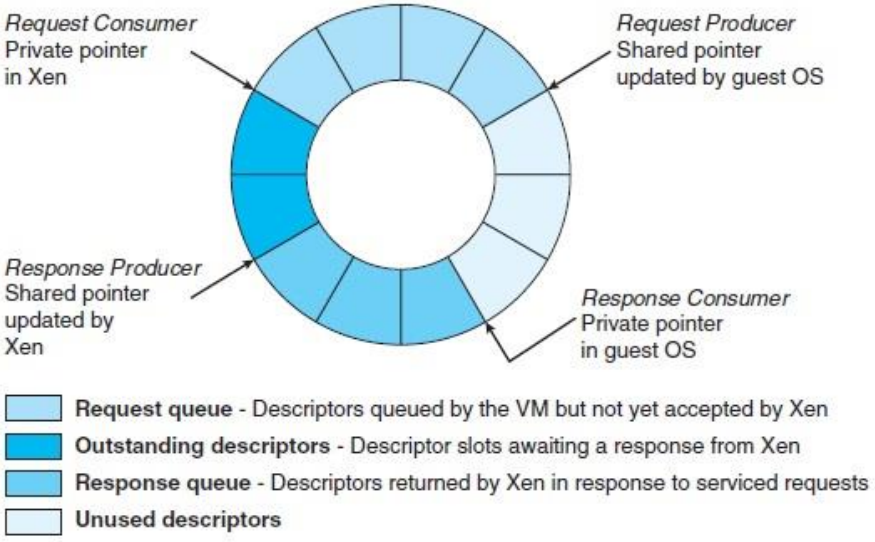


Figure 16.6 Xen I/O via shared circular buffer.

For memory management, Xen does not implement nested page tables.
Rather, each guest has its own set of page tables, set to read-only.
Xen requires the guest to use a specific mechanism, a hypercall from the guest to the hypervisor VMM, when a page-table change is needed.
This means that the guest operating system's kernel code must be changed from the default code to these Xen-specific methods.
To optimize performance, Xen allows the guest to queue up multiple page-table changes asynchronously via hypercalls and then check to ensure that the changes are complete before continuing operation.
Xen allowed virtualization of x86 CPUs without the use of binary translation, instead requiring modifications in the guest operating systems like the one described above.
Over time, Xen has taken advantage of hardware features supporting virtualization.
As a result, it no longer requires modified guests and essentially does not need the paravirtualization method.
Paravirtualization is still used in other solutions, however, such as type 0 hypervisors.

Programming-Environment Virtualization

Another kind of virtualization, based on a different execution model, is the virtualization of programming environments.
Here, a programming language is designed to run within a custom-built virtualized environment.
For example, Oracle's Java has many features that depend on its running in the Java virtual machine (JVM), including specific methods for security and memory management.

If we define virtualization as including only duplication of hardware, this is not really virtualization at all. But we need not limit ourselves to that definition.
Instead, we can define a virtual environment, based on APIs, that provides a set of features that we want to have available for a particular language and programs written in that language.
Java programs run within the JVM environment, and the JVM is compiled to be a native program on systems on which it runs. This arrangement means that Java programs are written once and then can run on any system (including all of the major operating systems) on which a JVM is available.
The same can be said for interpreted languages, which run inside programs that read each instruction and interpret it into native operations.

Emulation

Virtualization is probably the most common method for running applications designed for one operating system on a different operating system, but on the same CPU.
This method works relatively efficiently because the applications were compiled for the same instruction set as the target system uses.
Emulation is useful when the host system has one system architecture and the guest system was compiled for a different architecture.
For example, suppose a company has replaced its out dated computer system with a new system but would like to continue to run certain important programs that were compiled for the old system.
The programs could be run in an emulator that translates each of the out dated system's instructions into the native instruction set of the new system.
Emulation can increase the life of programs and allow us to explore old architectures without having an actual old machine.
As may be expected, the major challenge of emulation is performance.
Instruction-set emulation can run an order of magnitude slower than native instructions, because it may take ten instructions on the new system to read, parse, and simulate an instruction from the old system.
Thus, unless the new machine is ten times faster than the old, the program running on the new machine will run more slowly than it did on its native hardware.
Another challenge for emulator writers is that it is difficult to create a correct emulator because, in essence, this task involves writing an entire CPU in software.
In spite of these challenges, emulation is very popular, particularly in gaming circles.
Modern systems are so much faster than old game consoles that even the Apple iPhone has game emulators and games available to run within them.

Application Containment

The goal of virtualization in some instances is to provide a method to segregate applications, manage their performance and resource use, and create an easy way to start, stop, move, and manage them.
In such cases, perhaps full-fledged virtualization is not needed. If the applications are all compiled for the same operating system, then we do not need complete virtualization to provide these features. We can instead use application containment.
Consider one example of application containment. Starting with version 10, Oracle Solaris has included containers, or zones, that create a virtual layer between the operating system and the applications.

In this system, only one kernel is installed, and the hardware is not virtualized. Rather, the operating system and its devices are virtualized, providing processes within a zone with the impression that they are the only processes on the system.

One or more containers can be created, and each can have its own applications, network stacks, network address and ports, user accounts, and so on.

CPU and memory resources can be divided among the zones and the system-wide processes.

Each zone in fact can run its own scheduler to optimize the performance of its applications on the allotted resources. Figure 16.7 shows a Solaris 10 system with two containers and the standard "global" user space.

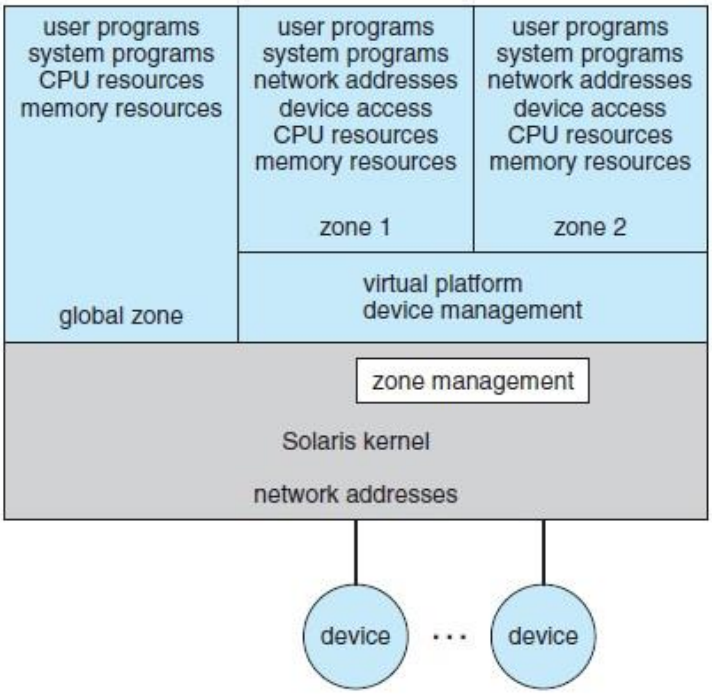


Figure 16.7 Solaris 10 with two zones.

Virtualization and Operating-System Components

Operating system aspects of virtualization, including how the VMM provides core operating-system functions like scheduling, I/O, and memory management.

How do VMMs schedule CPU use when guest operating systems believe they have dedicated CPUs?

How can memory management work when many guests require large amounts of memory?

CPU Scheduling

A system with virtualization, even a single-CPU system, frequently acts like a multiprocessor system.

The virtualization software presents one or more virtual CPUs to each of the virtual machines running on the system and then schedules the use of the physical CPUs among the virtual machines.

The VMM has a number of physical CPUs available and a number of threads to run on those CPUs. The threads can be VMM threads or guest threads.

Guests are configured with a certain number of virtual CPUs at creation time, and that number can be adjusted throughout the life of the VM.

When there are enough CPUs to allocate the requested number to each guest, the VMM can treat the CPUs as dedicated and schedule only a given guest's threads on that guest's CPUs.

In this situation, the guests act much like native operating systems running on native CPUs.
Of course, in other situations, there may not be enough CPUs to go around.
The VMM itself needs some CPU cycles for guest management and I/O management and can steal cycles from the guests by scheduling its threads across all of the system CPUs, but the impact of this action is relatively minor.
Commonly, the time-of-day clocks in virtual machines are incorrect because timers take longer to trigger than they would on dedicated CPUs. Virtualization can thus undo the good scheduling-algorithm efforts of the operating systems within virtual machines.
To correct for this, a VMM will have an application available for each type of operating system that system administrators install into the guests. This application corrects clock drift and can have other functions such as virtual device management.

Memory Management

Efficient memory use in general-purpose operating systems is one of the major keys to performance.
In virtualized environments, there are more users of memory (the guests and their applications, as well as the VMM), leading to more pressure on memory use.
Further adding to this pressure is that VMMs typically overcommit memory, so that the total memory with which guests are configured exceeds the amount of memory that physically exists in the system.
The extra need for efficient memory use is not lost on the implementers of VMMs, who take great measures to ensure the optimal use of memory.
For example, VMware ESX guests have a configured amount of physical memory, then ESX uses 3 methods of memory management

1. Double-paging, in which the guest page table indicates a page is in a physical frame but the VMM moves some of those pages to backing store
2. Install a pseudo-device driver in each guest (it looks like a device driver to the guest kernel but really just adds kernel-mode code to the guest)
    - Balloon memory manager communicates with VMM and is told to allocate or deallocate memory to decrease or increase physical memory use of guest, causing guest OS to free or have more memory available
3. Deduplication by VMM determining if same page loaded more than once, memory mapping the same page into multiple guests

I/O

Easier for VMMs to integrate with guests because I/O has lots of variation

- Already somewhat segregated / flexible via device drivers
- VMM can provide new devices and device drivers

But overall I/O is complicated for VMMs

- Many short paths for I/O in standard OSes for improved performance
- Less hypervisor needs to do for I/O for guests, the better
- Possibilities include direct device access, DMA pass-through, direct interrupt delivery
    - Again, HW support needed for these

Networking also complex as VMM and guests all need network access

- VMM can bridge guest to network (allowing direct access)

- And / or provide network address translation (NAT), NAT address local to machine on which guest is running, VMM provides address translation to guest to hide its address

Storage Management

- Both boot disk and general data access need be provided by VMM
- Need to support potentially dozens of guests per VMM (so standard disk partitioning not sufficient)
- **Type 1 –** storage guest root disks and config information within file system provided by VMM as a disk image
- **Type 2 –** store as files in file system provided by host OS
- **Duplicate file ->** create new guest
- Move file to another system -> move guest
- **Physical-to-virtual (P-to-V)** convert native disk blocks into VMM format
- **Virtual-to-physical (V-to-P)** convert from virtual format to native or disk format
- VMM also needs to provide access to network attached storage (just networking) and other disk images, disk partitions, disks, etc.,

Live Migration

Taking advantage of VMM features leads to new functionality not found on general operating systems such as live migration

Running guest can be moved between systems, without interrupting user access to the guest or its apps

Very useful for resource management, maintenance downtime windows, etc

1. The source VMM establishes a connection with the target VMM
2. The target creates a new guest by creating a new VCPU, etc
3. The source sends all read-only guest memory pages to the target
4. The source sends all read-write pages to the target, marking them as clean
5. The source repeats step 4, as during that step some pages were probably modified by the guest and are now dirty
6. When cycle of steps 4 and 5 becomes very short, source VMM freezes guest, sends VCPU's final state, sends other state details, sends final dirty pages, and tells target to start running the guest

 Once target acknowledges that guest running, source terminates guest

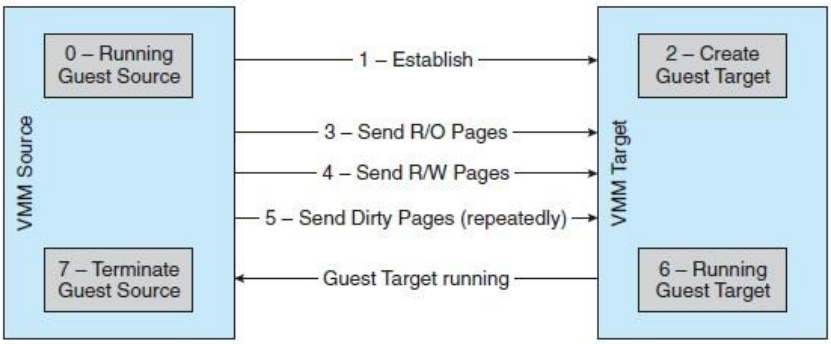


Figure 16.8 Live migration of a guest between two servers.

Mobile Operating System

A mobile operating system is an operating system that helps to run other application software on mobile devices. It is the same kind of software as the famous computer operating systems like Linux and Windows, but now they are light and simple to some extent.

The operating systems found on smartphones include Symbian OS, iPhone OS, RIM's BlackBerry, Windows Mobile, Palm WebOS, Android, and Maemo. Android, WebOS, and Maemo are all derived from Linux. The iPhone OS originated from BSD and NeXTSTEP, which are related to Unix.

It combines the beauty of computer and hand use devices. It typically contains a cellular built-in modem and SIM tray for telephony and internet connections. If you buy a mobile, the manufacturer company chooses the OS for that specific device.

Popular platforms of the Mobile OS

1. Android OS: The Android operating system is the most popular operating system today. It is a mobile OS based on the Linux Kernel and open-source software. The android operating system was developed by Google. The first Android device was launched in 2008.

2. Bada (Samsung Electronics): Bada is a Samsung mobile operating system that was launched in 2010. The Samsung wave was the first mobile to use the bada operating system. The bada operating system offers many mobile features, such as 3-D graphics, application installation, and multipoint-touch.

3. BlackBerry OS: The BlackBerry operating system is a mobile operating system developed by Research In Motion (RIM). This operating system was designed specifically for BlackBerry handheld devices. This operating system is beneficial for the corporate users because it provides synchronization with Microsoft Exchange, Novell GroupWise email, Lotus Domino, and other business software when used with the BlackBerry Enterprise Server.

4. iPhone OS / iOS: The iOS was developed by the Apple inc for the use on its device. The iOS operating system is the most popular operating system today. It is a very secure operating system. The iOS operating system is not available for any other mobiles.

5. Symbian OS: Symbian operating system is a mobile operating system that provides a high-level of integration with communication. The Symbian operating system is based on the java language. It combines middleware of wireless communications and personal information management (PIM) functionality. The Symbian operating system was developed by Symbian Ltd in 1998 for the use of mobile phones. Nokia was the first company to release Symbian OS on its mobile phone at that time.

6. Windows Mobile OS: The window mobile OS is a mobile operating system that was developed by Microsoft. It was designed for the pocket PCs and smart mobiles.

7. Harmony OS: The harmony operating system is the latest mobile operating system that was developed by Huawei for the use of its devices. It is designed primarily for IoT devices.

8. Palm OS: The palm operating system is a mobile operating system that was developed by Palm Ltd for use on personal digital assistants (PADs). It was introduced in 1996. Palm OS is also known as the Garnet OS.

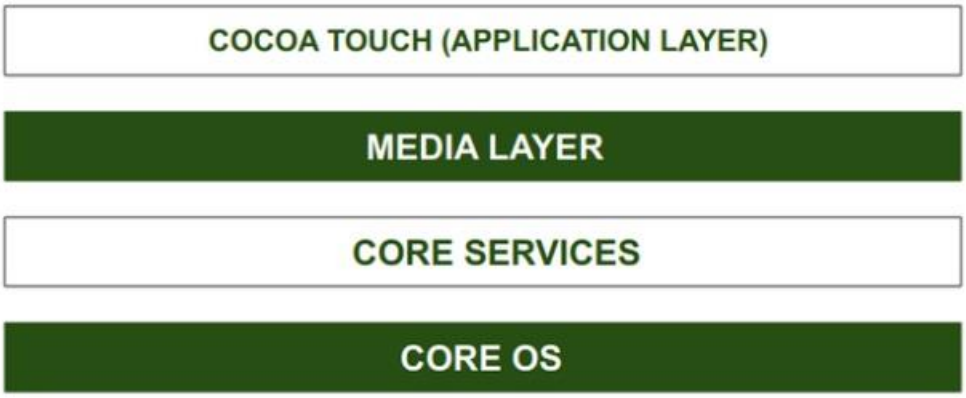
9. WebOS (Palm/HP): The WebOS is a mobile operating system that was developed by Palm. It based on the Linux Kernel. The HP uses this operating system in its mobile and touchpads.

Architecture of IOS

IOS is a Mobile Operating System that was developed by Apple Inc. for iPhones, iPads, and other Apple mobile devices. iOS is the second most popular and most used Mobile Operating System after Android.

The structure of the iOS operating System is Layered based. Its communication doesn't occur directly. The layer's between the Application Layer and the Hardware layer will help for Communication. The lower level gives basic services on which all applications rely and the higher-level layers provide graphics and interface-related services. Most of the system interfaces come with a special package called a framework.

A framework is a directory that holds dynamic shared libraries like .a files, header files, images, and helper apps that support the library. Each layer has a set of frameworks that are helpful for developers.



Architecture of IOS

CORE OS Layer:

All the IOS technologies are built under the lowest level layer i.e. Core OS layer. These technologies include:

- Core Bluetooth Framework
- External Accessories Framework
- Accelerate Framework
- Security Services Framework
- Local Authorization Framework etc.

It supports 64 bit which enables the application to run faster.

CORE SERVICES Layer:

Some important frameworks are present in the CORE SERVICES Layer which helps the iOS operating system to cure itself and provide better functionality. It is the 2nd lowest layer in the Architecture as shown above. Below are some important frameworks present in this layer:

Address Book Framework-

The Address Book Framework provides access to the contact details of the user.

Cloud Kit Framework-

This framework provides a medium for moving data between your app and iCloud.

Core Data Framework-

This is the technology that is used for managing the data model of a Model View Controller app.

Core Foundation Framework-

This framework provides data management and service features for iOS applications.

Core Location Framework-

This framework helps to provide the location and heading information to the application.

Core Motion Framework-

All the motion-based data on the device is accessed with the help of the Core Motion Framework.

Foundation Framework-

Objective C covering too many of the features found in the Core Foundation framework.

HealthKit Framework-

This framework handles the health-related information of the user.

HomeKit Framework-

This framework is used for talking with and controlling connected devices with the user's home.

Social Framework-

It is simply an interface that will access users' social media accounts.

StoreKit Framework-

This framework supports for buying of contents and services from inside iOS apps.

MEDIA Layer:

With the help of the media layer, we will enable all graphics video, and audio technology of the system. This is the second layer in the architecture. The different frameworks of MEDIA layers are:

ULKit Graphics-

This framework provides support for designing images and animating the view content.

Core Graphics Framework-

This framework support 2D vector and image-based rendering and it is a native drawing engine for iOS.

Core Animation-

This framework helps in optimizing the animation experience of the apps in iOS.

Media Player Framework-

This framework provides support for playing the playlist and enables the user to use their iTunes library.

AV Kit-

This framework provides various easy-to-use interfaces for video presentation, recording, and playback of audio and video.

Open AL-

This framework is an Industry Standard Technology for providing Audio.

Core Images-

This framework provides advanced support for motionless images.

GL Kit-

This framework manages advanced 2D and 3D rendering by hardware-accelerated interfaces.

COCOA TOUCH:

COCOA Touch is also known as the application layer which acts as an interface for the user to work with the iOS Operating system. It supports touch and motion events and many more features. The COCOA TOUCH layer provides the following frameworks:

EvenKit Framework-

This framework shows a standard system interface using view controllers for viewing and changing events.

GameKit Framework-

This framework provides support for users to share their game-related data online using a Game Center.

MapKit Framework-

This framework gives a scrollable map that one can include in your user interface of the app.

PushKit Framework-

This framework provides registration support.

Features of iOS operating System:

1. Highly Securer than other operating systems.
2. iOS provides multitasking features like while working in one application we can switch to another application easily.
3. iOS's user interface includes multiple gestures like swipe, tap, pinch, Reverse pinch.
4. iBooks, iStore, iTunes, Game Center, and Email are user-friendly.
5. It provides Safari as a default Web Browser.
6. It has a powerful API and a Camera.
7. It has deep hardware and software integration

Applications of IOS Operating System:

1. iOS Operating System is the Commercial Operating system of Apple Inc. and is popular for its security.
2. iOS operating system comes with pre-installed apps which were developed by Apple like Mail, Map, TV, Music, Wallet, Health, and Many More.
3. Swift Programming language is used for Developing Apps that would run on IOS Operating System.
4. In iOS Operating System we can perform Multitask like Chatting along with Surfing on the Internet.

Advantages of IOS Operating System:

1. More secure than other operating systems.
2. Excellent UI and fluid responsive
3. Suits best for Business and Professionals
4. Generate Less Heat as compared to Android.

Disadvantages of IOS Operating System:

1. More Costly.
2. Less User Friendly as Compared to Android Operating System.
3. Not Flexible as it supports only IOS devices.
4. Battery Performance is poor.

Android architecture

Android architecture contains different number of components to support any android device needs.

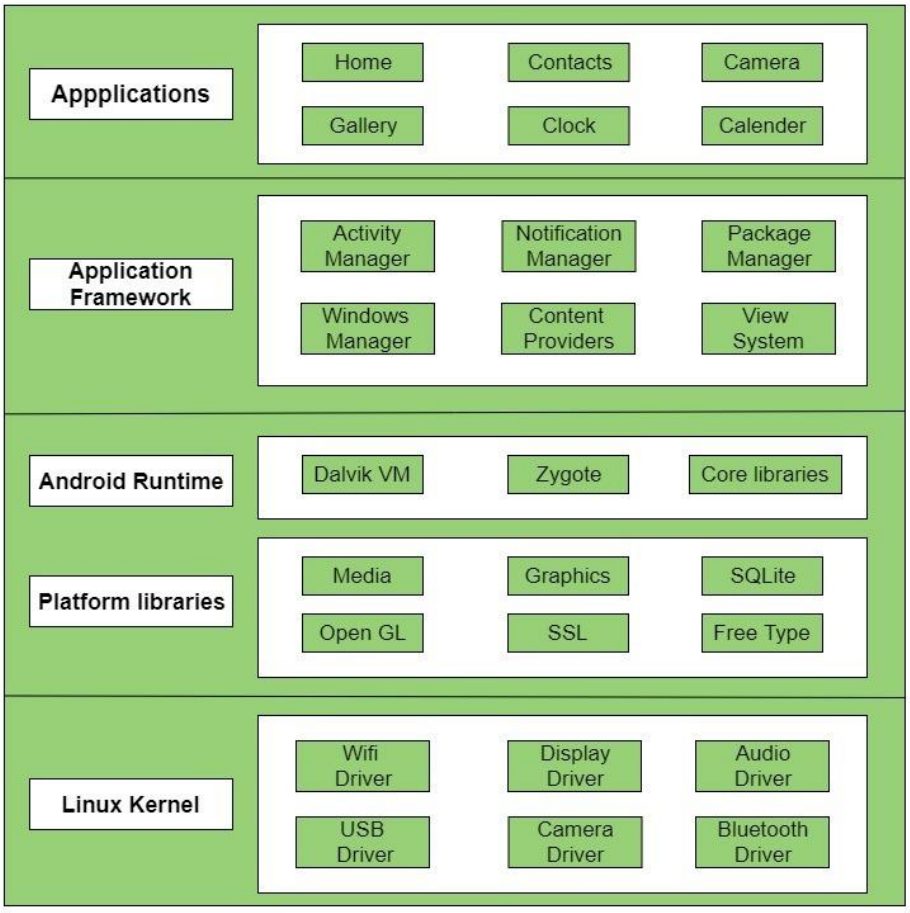
Android software contains an open-source Linux Kernel having collection of number of C/C++ libraries which are exposed through an application framework services.

Among all the components Linux Kernel provides main functionality of operating system functions to smartphones and Dalvik Virtual Machine (DVM) provide platform for running an android application.

The main components of android architecture are following:-

1. Applications
2. Application Framework
3. Android Runtime
4. Platform Libraries
5. Linux Kernel

Pictorial representation of android architecture with several main components and their sub components –



Applications

Applications is the top layer of android architecture. The pre-installed applications like home, contacts, camera, gallery etc and third party applications downloaded from the play store like chat applications, games etc. will be installed on this layer only.

It runs within the Android run time with the help of the classes and services provided by the application framework.

Application framework

Application Framework provides several important classes which are used to create an Android application. It provides a generic abstraction for hardware access and also helps in managing the user interface with application resources. Generally, it provides the services with the help of which we can create a particular class and make that class helpful for the Applications creation.

It includes different types of services activity manager, notification manager, view system, package manager etc. which are helpful for the development of our application according to the prerequisite.

Application runtime

Android Runtime environment is one of the most important part of Android. It contains components like core libraries and the Dalvik virtual machine (DVM). Mainly, it provides the base for the application framework and powers our application with the help of the core libraries.

Like Java Virtual Machine (JVM), Dalvik Virtual Machine (DVM) is a register-based virtual machine and specially designed and optimized for android to ensure that a device can run multiple instances efficiently. It depends on the layer Linux kernel for threading and low-level memory management. The core libraries enable us to implement android applications using the standard JAVA or Kotlin programming languages.

Platform libraries

The Platform Libraries includes various C/C++ core libraries and Java based libraries such as Media, Graphics, Surface Manager, OpenGL etc. to provide a support for android development.

Media library provides support to play and record an audio and video formats.

Surface manager responsible for managing access to the display subsystem.

SGL and OpenGL both cross-language, cross-platform application program interface (API) are used for 2D and 3D computer graphics.

SQLite provides database support and FreeType provides font support.

Web-Kit This open source web browser engine provides all the functionality to display web content and to simplify page loading.

SSL (Secure Sockets Layer) is security technology to establish an encrypted link between a web server and a web browser.

Linux Kernel

Linux Kernel is heart of the android architecture. It manages all the available drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc. which are required during the runtime. The Linux Kernel will provide an abstraction layer between the device hardware and the other components of android architecture. It is responsible for management of memory, power, devices etc.

The features of Linux kernel are:

Security: The Linux kernel handles the security between the application and the system.

Memory Management: It efficiently handles the memory management thereby providing the freedom to develop our apps.

Process Management: It manages the process well, allocates resources to processes whenever they need them.

Network Stack: It effectively handles the network communication.

Driver Model: It ensures that the application works properly on the device and hardware manufacturers responsible for building their drivers into the Linux build.