Software Engineering
ISSN: 2229-4007 & ISSN: 2229-4015, Volume 3, Issue 1, 2012, pp.-21-24.
Available online at http://www.bioinfo.in/contents.php?id=76

A ROLE OF VIRTUAL MACHINE IN OPERATING SYSTEM

KAWADE M.G.* AND BALLAL A.S.

Department of Information Technology, Jawaharlal Darda Institute of Engineering & Technology, Yavatmal, MS, India.
*Corresponding Author: Email- mkawade17@gmail.com, ameyb15@yahoo.com

Received: February 21, 2012; Accepted: March 15, 2012

Abstract- In this paper, we are concentrating about the role of Virtual Machine with Enviroment. virtual machine helps us to operate multiple OS on the same hardware. At the same time.Virtual machine can support individual processes or complete system.Depending on the abstraction level where virtualization occurs. The Operating System actually running on the hardware is called host operating system and the operating system running in the simulated enviroment is called guest operating system.
In this paper, we find that a few simple extension to a host os can make it a much faster platform for running a VM taking advantage of this extension reduce virtualization overhead for a virtual machine to 14-35% overhead even for workloads that exercise the virtual machine intensively.
Keywords- Virtual machine, Host OS, Guest OS.

Citation: Kawade M.G. and Ballal A.S.(2012) A Role of virtual machine in operating system. Software Engineering, ISSN: 2229-4007 & ISSN: 2229-4015, Volume 3, Issue 1, pp.-21-24.

Introduction of Operating System

The software that controls the computer and how it uses its resources is called the Operating System. This software manages and controls what happens in the computer operating system. The operating system provides two main functions. The first function is managing the basic hardware operations. The control of input and output, storage space, detecting equipment failure, and management of storage are just some of the responsibilities of the O/S or Operating System. The second function is managing and interacting with the applications software. It takes over the tasks of printing and saving data. To provide an environment for a computer user to execute programs.

Basic Functions of Operating System

Manage Storage Space-The operating system stores data at some location on disk. It knows where to go to retrieve data when it is needed. It uses the filing cabinet system to keep track of the data stored on disks, tape drives, CD-ROMS, and external drives.

Detects Problems and Equipment Failure-The operating system also is the maintenance mechanic of the system. It checks the system for failures that will cause problems in processing. Messages will appear on the screen when there is a problem. Sometimes operating systems will have built in messages for quick fixes to the problem, or will refer you to a resource to get more information. A typical message that one would see is “System Failure, or "Your computer has performed an illegal operation". When the computer is turned on, the computer checks all of the storage devices. You can see the system being checked by the lights going off and on at the various drive locations. If the computer cannot do a self fix, it will not let you continue working.

Traffic Controller-The operating system is also in charge of the data that is coming into the computer and going out of the computer. It directs the flow of data to and from the external devices

and also takes care of control routing information along the bus to be processed by the processor.

System Resource Manager- What is the system resource? Well any hardware or part of the computer used by the computer program is considered a system resource. (Talk about using someone.) The memory, disk drive, external devices, etc. are all "mothered" by the operating system. The O/S "allocates" or makes sure that enough space is there for the computer program to operate, allocates, time for each program to work, and also keeps the processor going after each instruction. Almost like a teacher standing over you to make sure you finish a problem.

Multitasking- Multitasking is the ability to run more than one program at a time. (You will find this feature in the Windows Operating System.) Multitasking will allow either the individual to work on more than one program at a time, or allow more than one user to share information and processing of the information. The O/S manages this operation.

Security Cop- Security on the system is also managed by the operating system. The O/S can give you the option to set up passwords or ID logons in order to use the computer. This provides security of data, and for those of us that don't want our parents to read what we have written.

Introduction to Virtual Operating System

The idea of a virtual operating system is to provide standard versions of the following-

1. Operating system primitives accessible through programming languages
2. The utility programs such as compilers, linkers and editors
3. The command language or means by which users access system resources from a terminal based on organizational requirements.

Virtual Memory

A Virtual Memory has the advantage of allowing more processes to run than the allowed memory size. This is achieved by only including parts of processes that are necessary to run in memory and the rest on disk. The absolute minimum part of a process that must always be in memory is called its working set. Usually, a program doesn't need to have its entire binary file in memory to run when it is performing a task that only uses part of its file. What this means is that, say, a 16MB program could happily run on a machine with only 4MB of memory.

Virtual Machine

A virtual machine is a completely isolated guest operating system installation within a normal host operating system. Virtual Machine Operating System creates illusion of multiple processors each capable of executing independently. Virtual machines are separated into two major categories, based on their use and degree of correspondence to any real machine.

System virtual machines

A system virtual machine provides a complete system platform which supports the execution of a complete operating system (OS).Multiple OS environments can co-exist on the same computer, in strong isolation from each other. The virtual machine can provide (ISA) that is somewhat different from that of the real machine. Application provisioning, maintenance, high availability and disaster recovery.

Process virtual machines

A process virtual machine is designed to run a single program, which means that it supports a single process. Its purpose is to provide a platform-independent programming environment that abstracts away details of the underlying hardware or operating system, and allows a program to execute in the same way on any platform.

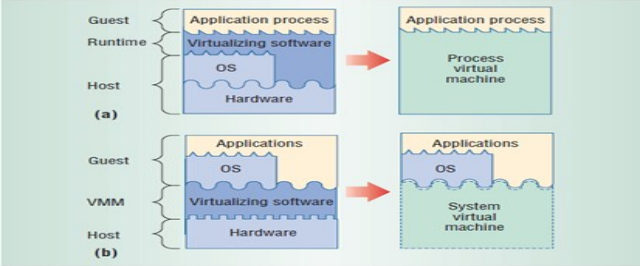


Fig.1- Process and system VMs. (a) In a process VM, virtualizing software translates a set of OS and user-level instructions composing one platform to those of another. (b) In a system VM, virtualizing software translates the ISA used by one hardware platform to that of another.

Working Of Virtual Machine OS

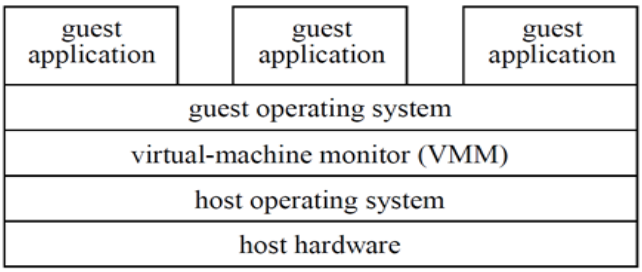


Fig.2- Virtual-machine structures.

A virtual-machine monitor is a software layer that runs on a host platform and provides an abstraction of a complete computer system to higher-level software. The software running above the virtual-machine abstraction is called guest software (operating system and applications).

Our goal for this paper is to examine and reduce the performance overhead associated with running a VMM on a host operating system. Building it on a standard Linux host operating system leads to an order of magnitude performance degradation compared to running outside a virtual machine (a standalone system). However, we find that a few simple extensions to the host operating system reduces virtualization overhead to 14-35% overhead, which is comparable to the speed of virtual machines that run directly on the hardware.

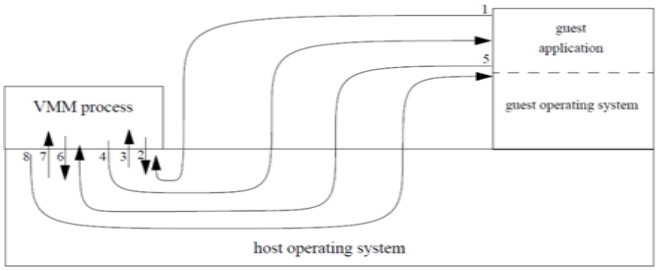


Fig. 3- Guest application system call.

This picture shows the steps UMLinux takes to transfer control to the guest operating system when a guest application process issues a system call. The mmap call in the SIGUSR1 handler must reside in guest user space. For security, the rest of the SIGUSR1 handler should reside in guest kernel space. The current UMLinux implementation includes an extra section of trampoline code to issue the mmap; this trampoline code is started by manipulating the guest machine process's context and finishes by causing a breakpoint to the VMM process; the VMM process then transfers control back to the guest-machine process by sending a SIGUSR1.

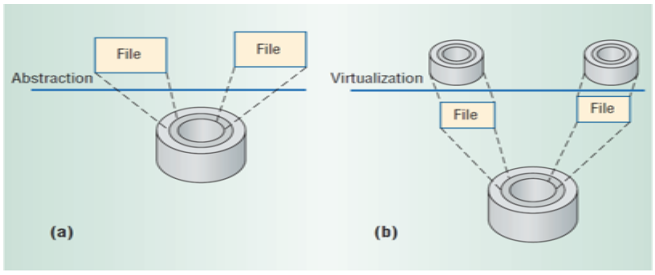
1. Guest application issues system call; intercepted by VMM process via ptrace
2. VMM process changes system call to no-op (getpid)
3. Getpid returns; intercepted by VMM process
4. VMM process sends SIGUSR1 signal to guest SIGUSR1 handler
5. Guest SIGUSR1 handler calls mmap to allow access to guest kernel data; intercepted by VMM process
6. VMM process allows mmap to pass through
7. mmap returns to VMM process
8. VMM process returns to guest SIGUSR1 handler, which hadles the guest application's system call.

Architecture of Virtual Machine

Abstraction and Virtualization

Despite their incredible complexity, computer systems exist and continue to evolve because they are designed as hierarchies with well-defined interfaces that separate levels of abstraction. Using well-defined interfaces facilitates independent subsystem development by both hardware and software design teams. The simplifying abstractions hide lower-level implementation details, thereby reducing the complexity of the design process. Figure 1a, shows an example of abstraction applied to disk storage. The operating system abstracts hard-disk addressing details—for example, that it is comprised of sectors and tracks—so that the disk appears to application software as a set of variable-sized files. Application programmer scan then create, write, and read files without knowing the hard disk's construction and physical organization.

Unlike abstraction, virtualization does not necessarily aim to simplify or hide details. For example, in Figure1b, virtualization transforms a single large disk into two smaller virtual disks, each of which appears to have its own tracks and sectors. Virtualization software uses the file abstraction as an intermediate step to provide a mapping between the virtual and real disks. A write to a virtual disk is converted to a file write (and therefore to a real disk write). Note that the level of detail provided at the virtual disk interface—the sector/track addressing— is no different from that for a real

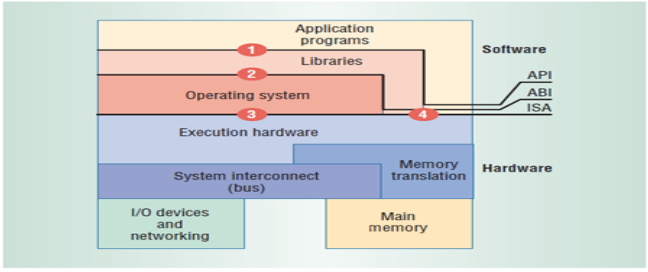


disk; no abstraction takes place.

Fig.4- Abstraction and virtualization applied to disk storage.(a) Abstraction provides a simplified interface to underlying resources. (b) Virtualization provides a different interface or different resources at the same abstraction level.

Architected interfaces-

Fig. 5 shows some important interfaces and implementation layers in a typical computer system. Three of these interfaces at or near the HW/SW boundary—the instruction set architecture, the application binary interface, and the application programming interface—



are especially important for VM construction.

Fig.5- Computer system architecture. Key implementation layers communicate vertically via the instruction set architecture (ISA), application binary interface (ABI), and application programming interface (API).

Instruction set architecture

The ISA marks the division between hardware and software, and consists of interfaces 3 and 4 in Fig. 2 Interface 4 represents the user ISA and includes those aspects visible to an application program. Interface 3, the system ISA, is a superset of the user ISA and includes those aspects visible only to operating system software responsible for managing hardware resources.

Application binary interface

The ABI gives a program access to the hardware resources and services available in a system through the user ISA (interface 4) and the system call interface (interface 2). The ABI does not include system instructions; rather, all application programs interact with the hardware resources indirectly by invoking the operating system's services via the system call interface. System calls provide a way for an operating system to perform operations on behalf of a user program after validating their authenticity and safety.

Application programming interface
The API gives a program access to the hardware resources and services available in a system through the user ISA (interface 4) supplemented with high-level language (HLL) library calls (interface 1). Any system calls are usually performed through libraries. Using an API enables application software to be ported easily, through recompilation, to other systems that support the same API.

Advantages
1. Multiple OS environments can exist simultaneously on the same machine, isolated from each other.
2. Virtual machine can offer an instruction set architecture that differs from real computer's.
3. Easy maintenance, application provisioning, availability and convenient recovery.

Disadvantages
1. Virtual machine is not that efficient as a real one when accessing the hardware.
2. When multiple virtual machines are simultaneously running on a host computer, each virtual machine may introduce an unstable performance, which depends on the workload on the system by other running virtual machines.
3. The host OS required a separate host user process to control the main guest-machine process, and this generated a large number of host context switches.

Applications
Try new operating systems- If you want to try a new OS on same hardware. Put together a VM and build Ubuntu on it. Suddenly, you can launch and try dozens of operating systems without much hassle.

Test your software- When we create new software like web application, application program etc. we can test that software on different OS by using the VMware. Weather the software run efficiently on different OS or not.

Small Biz disaster recovery- This isn't highly recommended, but it'd work if you're bootstrapping. Say you're hosting a few web servers with your amazing app on them. Your house gets hit by lightning. Your site is off the air. Now, imagine that scenario but you've got virtual backups of the latest build and configuration ready to install and deploy wherever else you've got a point of presence. Poof. You're online again.

Build kid boxes- Build Edubuntu (a kid flavored Ubuntu) on a virtual machine for the kids (the specs I mention above are for heavy users, but you could get away with a lot less if you only ran ONE VM). If (when) things go sour from one too many "tweaks," just drop the VM and restore from your pristine copy. Talk about easy. You can get them back on the net in less than 10 minutes.

Backup your system- When you get ready to move from XP to Vista, you can use VMware to make a backup of your old system. If things go horribly sour, you could have the VM version up and running in short order. By the way, you can have TWO servers, and have a copy of the VM on both. This would give you even more business continuity, should something happen to the server.

Save Legacy Systems- Offices and data centers often have an old box around that just can't be mucked with. There's additional software you can use to do what's called a P2V switch, a physical-to-virtual conversion, where the old box's "image" gets copied onto the virtual machine files, and thus, gives you a hopefully-operational clone of the old grandpa box in the corner.

Conclusion and Future Work
In the future, there can be reduction in the size of the host operating system used to support a VMM. Much of the code in the host OS can be eliminated, because the VMM uses only a small number of system calls and abstractions in the host OS. Reducing the code size of the host OS will help make VM a fast and trusted base for future virtual-machine services.
In this presentation we examine Virtual Machine on the basis of operating system. We study how there is actual working between Operating system and virtual machine and how we can access multiple operating system on same hardware.

References
[1] *The Bell Syst. Tech.* J. (1978) 57, 6.
[2] Bourne S.R. (1978) *The Bell Syst. Tech. J.* 57(6), 1971-1990.
[3] Hall D., Scherrer D. and Sventek J., *The software tools programmers manual*.
[4] Brinch Hansen P. (1973) *Operating System Principles*.
[5] Kernighan. *Software Practice and* Experience, 5 (4), 395-406.
[6] Kernighan B., and Plauger P. (1976) *Software Tools*.
[7] Scherrer D. *Instructions for implementing the LBL software tools package*.
[8] Richards. (1971) *Software-Practice and* Experience, 1 (2), 135 -146.